ISYE 6501 Homework 11

Question 15.1

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.

Part 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!)

Install required packages

In [13]:
```
#!pip install pulp
#!pip install xlrd
import pip
import pandas as pd
from pulp import *
import math
```

Read Excel File

In [16]:
```
#read excel file
excel_file = r"diet.xls"
diet_data = pd.read_excel(excel_file)[0:64]

#head data
print(diet_data.head())
```

```
                 Foods  Price/ Serving       Serving Size  Calories  \
0      Frozen Broccoli            0.16          10 Oz Pkg      73.8   
1          Carrots,Raw            0.07  1/2 Cup Shredded      23.7   
2          Celery, Raw            0.04            1 Stalk       6.4   
3          Frozen Corn            0.18            1/2 Cup      72.2   
4  Lettuce,Iceberg,Raw            0.02             1 Leaf       2.6   

   Cholesterol mg  Total_Fat g  Sodium mg  Carbohydrates g  Dietary_Fiber g  \
0             0.0          0.8       68.2             13.6              8.5   
1             0.0          0.1       19.2              5.6              1.6   
2             0.0          0.1       34.8              1.5              0.7   
3             0.0          0.6        2.5             17.1              2.0   
4             0.0          0.0        1.8              0.4              0.3   

   Protein g  Vit_A IU  Vit_C IU  Calcium mg  Iron mg  
0        8.0    5867.4     160.2       159.0      2.3  
1        0.6   15471.0       5.1        14.9      0.3  
2        0.3      53.6       2.8        16.0      0.2  
3        2.5     106.6       5.2         3.3      0.3  
4        0.2      66.0       0.8         3.8      0.1  
```

Step 1: define optimization problem

In [19]:
```
problem = LpProblem("Diet_Problem", LpMinimize)
#print(problem)
```

Step 2: define decision variables

In [22]:
```
#extract foods from excel file
foods = diet_data["Foods"].tolist()
#print(foods)

#create food variables
food_variables = {}
for food in foods:
    food_variables[food] = LpVariable(food, lowBound=0)
#print(food_variables)
```

Step 3: define objective function

In [25]:
```
#extract costs from excel file
costs = diet_data["Price/ Serving"].tolist()

#create a dictionary with foods and associated costs
food_costs = dict(zip(foods, costs))
#print(food_costs)

#create objective to minimize cost and add it to my problem
total_cost = lpSum([food_variables[food] * cost for food, cost in food_costs.items()])

#add objective function to problem
problem += total_cost
#print(problem)
```

Step 4: define nutritional constraints

In [28]:
```
#calories constraint
calories = diet_data["Calories"].tolist()
food_calories = dict(zip(foods, calories))
calorie_constraint_lower = lpSum([food_variables[food] * calories for food, calories i
calorie_constraint_upper = lpSum([food_variables[food] * calories for food, calories i
problem += calorie_constraint_lower
problem += calorie_constraint_upper

#cholesterol constraint
cholesterol = diet_data["Cholesterol mg"].tolist()
food_cholesterol = dict(zip(foods, cholesterol))
cholesterol_constraint_lower = lpSum([food_variables[food] * cholesterol for food, cho
cholesterol_constraint_upper = lpSum([food_variables[food] * cholesterol for food, cho
problem += cholesterol_constraint_lower
problem += cholesterol_constraint_upper

#fat constraint
fat = diet_data["Total_Fat g"].tolist()
food_fat = dict(zip(foods, fat))
fat_constraint_lower = lpSum([food_variables[food] * fat for food, fat in food_fat.ite
fat_constraint_upper = lpSum([food_variables[food] * fat for food, fat in food_fat.ite
problem += fat_constraint_lower
problem += fat_constraint_upper

#sodium constraint
sodium = diet_data["Sodium mg"].tolist()
food_sodium = dict(zip(foods, sodium))
sodium_constraint_lower = lpSum([food_variables[food] * sodium for food, sodium in foo
sodium_constraint_upper = lpSum([food_variables[food] * sodium for food, sodium in foo
problem += sodium_constraint_lower
problem += sodium_constraint_upper

#carbs constraint
carbs = diet_data["Carbohydrates g"].tolist()
food_carbs = dict(zip(foods, carbs))
carbs_constraint_lower = lpSum([food_variables[food] * carbs for food, carbs in food_c
carbs_constraint_upper = lpSum([food_variables[food] * carbs for food, carbs in food_c
problem += sodium_constraint_lower
problem += sodium_constraint_upper

#fiber constraint
fiber = diet_data["Dietary_Fiber g"].tolist()
food_fiber = dict(zip(foods, fiber))
fiber_constraint_lower = lpSum([food_variables[food] * fiber for food, fiber in food_f
fiber_constraint_upper = lpSum([food_variables[food] * fiber for food, fiber in food_f
problem += fiber_constraint_lower
problem += fiber_constraint_upper

#protein constraint
protein = diet_data["Protein g"].tolist()
food_protein = dict(zip(foods, protein))
protein_constraint_lower = lpSum([food_variables[food] * protein for food, protein in
protein_constraint_upper = lpSum([food_variables[food] * protein for food, protein in
problem += protein_constraint_lower
problem += protein_constraint_upper

#vitamin A constraint
vitA = diet_data["Vit_A IU"].tolist()
food_vitA = dict(zip(foods, vitA))
```

```
vitA_constraint_lower = lpSum([food_variables[food] * vitA for food, vitA in food_vitA
vitA_constraint_upper = lpSum([food_variables[food] * vitA for food, vitA in food_vitA
problem += vitA_constraint_lower
problem += vitA_constraint_upper

#vitamin C constraint
vitC = diet_data["Vit_C IU"].tolist()
food_vitC = dict(zip(foods, vitC))
vitC_constraint_lower = lpSum([food_variables[food] * vitC for food, vitC in food_vitC
vitC_constraint_upper = lpSum([food_variables[food] * vitC for food, vitC in food_vitC
problem += vitC_constraint_lower
problem += vitC_constraint_upper

#calcium constraint
calcium = diet_data["Calcium mg"].tolist()
food_calcium = dict(zip(foods, calcium))
calcium_constraint_lower = lpSum([food_variables[food] * calcium for food, calcium in
calcium_constraint_upper = lpSum([food_variables[food] * calcium for food, calcium in
problem += calcium_constraint_lower
problem += calcium_constraint_upper

#iron constraint
iron = diet_data["Iron mg"].tolist()
food_iron = dict(zip(foods, iron))
iron_constraint_lower = lpSum([food_variables[food] * iron for food, iron in food_iron
iron_constraint_upper = lpSum([food_variables[food] * iron for food, iron in food_iron
problem += iron_constraint_lower
problem += iron_constraint_upper
```

Step 5: solve optimization problem

In [31]:
```
problem.solve()
```

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

command line - /home/468b4343-99bb-473f-a468-4883e72eb3f7/.local/lib/python3.11/site-
packages/pulp/solverdir/cbc/linux/64/cbc /tmp/e90a63749a0b43a7aaf0af6873ca8c4d-pulp.m
ps timeMode elapsed branch printingOptions all solution /tmp/e90a63749a0b43a7aaf0af68
73ca8c4d-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 27 COLUMNS
At line 1292 RHS
At line 1315 BOUNDS
At line 1316 ENDATA
Problem MODEL has 22 rows, 64 columns and 1200 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 21 (-1) rows, 64 (0) columns and 1138 (-62) elements
0  Obj 0 Primal inf 21.034603 (11)
9  Obj 2.8984763
Optimal - objective value 2.8984763
After Postsolve, objective 2.8984763, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 2.898476251 - 9 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.03
```

Out[31]: 1

Step 6: retrieve optimal solution

In [34]:
```
#how much of each variable?
optimal_solution = {}
for variable in food_variables:
    #print(value(food_variables[variable]))
    if value(food_variables[variable]) != 0:
        optimal_solution[variable] = value(food_variables[variable])

#what is the cost?
optimal_objective = value(problem.objective)

#print solution
print("Optimal Solution:")
for food, amount in optimal_solution.items():
    print(food, ": ", amount)
print()
print("Lowest Cost:")
print("$", round(optimal_objective, 2))
```

```
Optimal Solution:
Frozen Broccoli :  1.0510324
Celery, Raw :  43.207217
Oranges :  1.5874377
Poached Eggs :  0.14184397
Popcorn,Air-Popped :  18.81398

Lowest Cost:
$ 2.9
```

Part 2

Please add to your model the following constraints (which might require adding more variables) and solve the new model:

Constraint 1: 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.)

Constraint 2: Many people dislike celery and frozen broccoli. So at most one, but not both, can be selected.

Constraint 3: 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!

Step 1: define optimization problem

In [186…
```
problem2 = LpProblem("Diet_Problem", LpMinimize)
#print(problem2)
```

Step 2: define decision variables

In [189…
```
#extract foods from excel file
foods = diet_data["Foods"].tolist()
#print(foods)

#create food variables
food_variables = {}
for food in foods:
    food_variables[food] = LpVariable(food, lowBound=0)
#print(food_variables)

#create binary food variables
food_variables_binary = {}
for food in foods:
    food_variables_binary[food] = LpVariable(f"{food}_binary", cat="Binary")
#print(food_variables_binary)
```

Step 3: define objective function

In [192…
```
#extract costs from excel file
costs = diet_data["Price/ Serving"].tolist()

#create a dictionary with foods and associated costs
food_costs = dict(zip(foods, costs))
#print(food_costs)

#create objective to minimize cost and add it to my problem
total_cost = lpSum([food_variables[food] * cost for food, cost in food_costs.items()])

#add objective function to problem
problem2 += total_cost
#print(problem)
```

Step 4a: define nutritional constraints

In [195…
```
#calories constraint
calories = diet_data["Calories"].tolist()
food_calories = dict(zip(foods, calories))
calorie_constraint_lower = lpSum([food_variables[food] * calories for food, calories i
calorie_constraint_upper = lpSum([food_variables[food] * calories for food, calories i
problem2 += calorie_constraint_lower
problem2 += calorie_constraint_upper

#cholesterol constraint
cholesterol = diet_data["Cholesterol mg"].tolist()
food_cholesterol = dict(zip(foods, cholesterol))
cholesterol_constraint_lower = lpSum([food_variables[food] * cholesterol for food, chc
cholesterol_constraint_upper = lpSum([food_variables[food] * cholesterol for food, chc
problem2 += cholesterol_constraint_lower
problem2 += cholesterol_constraint_upper

#fat constraint
fat = diet_data["Total_Fat g"].tolist()
```

```
food_fat = dict(zip(foods, fat))
fat_constraint_lower = lpSum([food_variables[food] * fat for food, fat in food_fat.ite
fat_constraint_upper = lpSum([food_variables[food] * fat for food, fat in food_fat.ite
problem2 += fat_constraint_lower
problem2 += fat_constraint_upper

#sodium constraint
sodium = diet_data["Sodium mg"].tolist()
food_sodium = dict(zip(foods, sodium))
sodium_constraint_lower = lpSum([food_variables[food] * sodium for food, sodium in foo
sodium_constraint_upper = lpSum([food_variables[food] * sodium for food, sodium in foo
problem2 += sodium_constraint_lower
problem2 += sodium_constraint_upper

#carbs constraint
carbs = diet_data["Carbohydrates g"].tolist()
food_carbs = dict(zip(foods, carbs))
carbs_constraint_lower = lpSum([food_variables[food] * carbs for food, carbs in food_c
carbs_constraint_upper = lpSum([food_variables[food] * carbs for food, carbs in food_c
problem2 += sodium_constraint_lower
problem2 += sodium_constraint_upper

#fiber constraint
fiber = diet_data["Dietary_Fiber g"].tolist()
food_fiber = dict(zip(foods, fiber))
fiber_constraint_lower = lpSum([food_variables[food] * fiber for food, fiber in food_f
fiber_constraint_upper = lpSum([food_variables[food] * fiber for food, fiber in food_f
problem2 += fiber_constraint_lower
problem2 += fiber_constraint_upper

#protein constraint
protein = diet_data["Protein g"].tolist()
food_protein = dict(zip(foods, protein))
protein_constraint_lower = lpSum([food_variables[food] * protein for food, protein in
protein_constraint_upper = lpSum([food_variables[food] * protein for food, protein in
problem2 += protein_constraint_lower
problem2 += protein_constraint_upper

#vitamin A constraint
vitA = diet_data["Vit_A IU"].tolist()
food_vitA = dict(zip(foods, vitA))
vitA_constraint_lower = lpSum([food_variables[food] * vitA for food, vitA in food_vitA
vitA_constraint_upper = lpSum([food_variables[food] * vitA for food, vitA in food_vitA
problem2 += vitA_constraint_lower
problem2 += vitA_constraint_upper

#vitamin C constraint
vitC = diet_data["Vit_C IU"].tolist()
food_vitC = dict(zip(foods, vitC))
vitC_constraint_lower = lpSum([food_variables[food] * vitC for food, vitC in food_vitC
vitC_constraint_upper = lpSum([food_variables[food] * vitC for food, vitC in food_vitC
problem2 += vitC_constraint_lower
problem2 += vitC_constraint_upper

#calcium constraint
calcium = diet_data["Calcium mg"].tolist()
food_calcium = dict(zip(foods, calcium))
calcium_constraint_lower = lpSum([food_variables[food] * calcium for food, calcium in
calcium_constraint_upper = lpSum([food_variables[food] * calcium for food, calcium in
problem2 += calcium_constraint_lower
```

```
problem2 += calcium_constraint_upper

#iron constraint
iron = diet_data["Iron mg"].tolist()
food_iron = dict(zip(foods, iron))
iron_constraint_lower = lpSum([food_variables[food] * iron for food, iron in food_iror
iron_constraint_upper = lpSum([food_variables[food] * iron for food, iron in food_iror
problem2 += iron_constraint_lower
problem2 += iron_constraint_upper
```

Step 4b: Define additional constraints

In [198…

```
#constraint 1
for food in foods:
    problem2 += food_variables[food] <= 10000 * food_variables_binary[food]
    problem2 += food_variables[food] >= 0.1 * food_variables_binary[food]

#constraint 2
problem2 += food_variables_binary["Frozen Broccoli"] + food_variables_binary["Celery,

#constraint 3
problem2 += food_variables_binary['Roasted Chicken'] + food_variables_binary['Poached
  food_variables_binary['Scrambled Eggs'] + food_variables_binary['Frankfurter, Beef']
  food_variables_binary['Kielbasa,Prk'] + food_variables_binary['Hamburger W/Toppings'
  food_variables_binary['Hotdog, Plain'] + food_variables_binary['Pork'] + \
  food_variables_binary['Bologna,Turkey'] + food_variables_binary['Ham,Sliced,Extralea
  food_variables_binary['White Tuna in Water'] \
  >= 3

#food_variables_binary["Tofu"] + food_variables_binary["Roasted Chicken"] + food_varia


#constraint3 = lpSum([food_variables_binary[food] for food in foods if str(food_variab
#problem2 += constraint3
```

Step 5: solve optimization problem

In [201…

```
problem2.solve()
```

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

command line - /home/468b4343-99bb-473f-a468-4883e72eb3f7/.local/lib/python3.11/site-
packages/pulp/solverdir/cbc/linux/64/cbc /tmp/a8c94b8da0ed4702b29e14ebdcb221fe-pulp.m
ps timeMode elapsed branch printingOptions all solution /tmp/a8c94b8da0ed4702b29e14eb
dcb221fe-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 157 COLUMNS
At line 1819 RHS
At line 1972 BOUNDS
At line 2037 ENDATA
Problem MODEL has 152 rows, 128 columns and 1469 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 2.95664 - 0.00 seconds
Cgl0003I 0 fixed, 0 tightened bounds, 64 strengthened rows, 0 substitutions
Cgl0004I processed model has 140 rows, 128 columns (64 integer (64 of which binary))
and 807 elements
Cbc0038I Initial state - 7 integers unsatisfied sum - 1.82862
Cbc0038I Pass   1: suminf.    0.06148 (1) obj. 4.57147 iterations 91
Cbc0038I Solution found of 4.57147
Cbc0038I Relaxing continuous gives 4.56434
Cbc0038I Before mini branch and bound, 56 integers at bound fixed and 54 continuous
Cbc0038I Full problem 140 rows 128 columns, reduced to 28 rows 18 columns
Cbc0038I Mini branch and bound improved solution from 4.56434 to 3.45406 (0.02 second
s)
Cbc0038I Round again with cutoff of 3.428
Cbc0038I Pass   2: suminf.    0.19368 (3) obj. 3.428 iterations 16
Cbc0038I Pass   3: suminf.    1.14843 (3) obj. 3.428 iterations 17
Cbc0038I Pass   4: suminf.    0.07496 (2) obj. 3.428 iterations 3
Cbc0038I Pass   5: suminf.    0.71766 (2) obj. 3.428 iterations 14
Cbc0038I Pass   6: suminf.    1.50676 (6) obj. 3.428 iterations 21
Cbc0038I Pass   7: suminf.    1.23934 (4) obj. 3.428 iterations 6
Cbc0038I Pass   8: suminf.    0.94259 (2) obj. 3.428 iterations 9
Cbc0038I Pass   9: suminf.    0.66890 (2) obj. 3.428 iterations 5
Cbc0038I Pass  10: suminf.    1.13422 (3) obj. 3.428 iterations 27
Cbc0038I Pass  11: suminf.    1.13422 (3) obj. 3.428 iterations 7
Cbc0038I Pass  12: suminf.    0.87867 (2) obj. 3.428 iterations 5
Cbc0038I Pass  13: suminf.    0.68593 (2) obj. 3.428 iterations 3
Cbc0038I Pass  14: suminf.    1.17906 (4) obj. 3.428 iterations 35
Cbc0038I Pass  15: suminf.    1.17906 (4) obj. 3.428 iterations 15
Cbc0038I Pass  16: suminf.    0.87293 (2) obj. 3.428 iterations 7
Cbc0038I Pass  17: suminf.    0.67381 (2) obj. 3.428 iterations 3
Cbc0038I Pass  18: suminf.    0.92821 (3) obj. 3.428 iterations 14
Cbc0038I Pass  19: suminf.    0.92821 (3) obj. 3.428 iterations 7
Cbc0038I Pass  20: suminf.    0.88070 (2) obj. 3.428 iterations 6
Cbc0038I Pass  21: suminf.    0.67675 (2) obj. 3.428 iterations 3
Cbc0038I Pass  22: suminf.    1.37974 (5) obj. 3.428 iterations 18
Cbc0038I Pass  23: suminf.    1.11994 (3) obj. 3.428 iterations 9
Cbc0038I Pass  24: suminf.    0.87067 (2) obj. 3.428 iterations 18
Cbc0038I Pass  25: suminf.    0.69631 (2) obj. 3.428 iterations 4
Cbc0038I Pass  26: suminf.    1.04414 (4) obj. 3.428 iterations 12
Cbc0038I Pass  27: suminf.    0.80463 (2) obj. 3.428 iterations 24
Cbc0038I Pass  28: suminf.    0.80522 (2) obj. 3.428 iterations 4
Cbc0038I Pass  29: suminf.    1.41367 (4) obj. 3.428 iterations 13
Cbc0038I Pass  30: suminf.    1.41367 (4) obj. 3.428 iterations 6
Cbc0038I Pass  31: suminf.    0.85670 (2) obj. 3.428 iterations 16
```

```
Cbc0038I No solution found this major pass
Cbc0038I Before mini branch and bound, 36 integers at bound fixed and 35 continuous
Cbc0038I Full problem 140 rows 128 columns, reduced to 68 rows 57 columns
Cbc0038I Mini branch and bound did not improve solution (0.03 seconds)
Cbc0038I After 0.04 seconds - Feasibility pump exiting with objective of 3.45406 - to
ok 0.02 seconds
Cbc0012I Integer solution of 3.4540591 found by feasibility pump after 0 iterations a
nd 0 nodes (0.04 seconds)
Cbc0038I Full problem 140 rows 128 columns, reduced to 26 rows 16 columns
Cbc0031I 4 added rows had average density of 16
Cbc0013I At root node, 22 cuts changed objective from 3.1935822 to 3.4540591 in 1 pas
ses
Cbc0014I Cut generator 0 (Probing) - 1 row cuts average 2.0 elements, 2 column cuts
(2 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 6 row cuts average 25.7 elements, 0 column cuts
(0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 1 row cuts average 23.0 elements, 0 column cuts
(0 active)  in 0.000 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) - 7 row cuts average 64.6 elements,
0 column cuts (0 active)  in 0.000 seconds - new frequency is 1
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) - 7 row cuts average 22.9 elements, 0 column cu
ts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0001I Search completed - best objective 3.454059119482558, took 0 iterations and 0
nodes (0.04 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from 3.19358 to 3.45406
Probing was tried 1 times and created 3 cuts of which 0 were active after adding roun
ds of cuts (0.000 seconds)
Gomory was tried 1 times and created 6 cuts of which 0 were active after adding round
s of cuts (0.000 seconds)
Knapsack was tried 1 times and created 1 cuts of which 0 were active after adding rou
nds of cuts (0.000 seconds)
Clique was tried 1 times and created 0 cuts of which 0 were active after adding round
s of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 1 times and created 7 cuts of which 0 were active aft
er adding rounds of cuts (0.000 seconds)
FlowCover was tried 1 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.000 seconds)
TwoMirCuts was tried 1 times and created 7 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding rou
nds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                3.45405912
Enumerated nodes:               0
Total iterations:               0
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
```

Out[201]: 1

Step 6: retrieve optimal solution

In [204…

```
#how much of each variable?
optimal_solution = {}
for variable in food_variables:
    #print(value(food_variables[variable]))
    if value(food_variables[variable]) != 0:
        optimal_solution[variable] = value(food_variables[variable])

#what is the cost?
optimal_objective = value(problem2.objective)

#print solution
print("Optimal Solution:")
for food, amount in optimal_solution.items():
    print(food, ": ", amount)
print()
print("Lowest Cost:")
print("$", round(optimal_objective, 2))
```

```
Optimal Solution:
Celery, Raw :  54.573459
Oranges :  3.5464034
Poached Eggs :  0.1
Scrambled Eggs :  0.1
Bologna,Turkey :  0.1
Popcorn,Air-Popped :  17.629007

Lowest Cost:
$ 3.45
```