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.]

```
!pip install pulp
!pip install pandas
```

First read the dataset and create problem variable.

```
from pulp import *
prob = LpProblem("Army_Diet_Problem",LpMinimize)
import pandas as pd
df = pd.read_excel("diet.xls",nrows=64, header=0)
df
```

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

64 rows × 14 columns

At the start of problem formulation, create the hash tables to store the key-value pairs for each column in the dataset.

```
# Create a list of the food items
food_items = list(df['Foods'])
```

```
# Create a dictinary of costs for all food items
costs = dict(zip(food_items,df['Price per Serving']))

# Create a dictinary of serving size for all food items
serving_size = dict(zip(food_items,df['Serving Size']))

# Create a dictionary of calories for all food items per serving size
calories = dict(zip(food_items,df['Calories']))

# Create a dictionary of calories for all food items
cholesterol = dict(zip(food_items,df['Cholesterol mg']))

# Create a dictionary of total fat for all food items
fat = dict(zip(food_items,df['Total_Fat g']))

# Create a dictionary of sodium for all food items
sodium = dict(zip(food_items,df['Sodium mg']))

# Create a dictionary of carbohydrates for all food items
carbs = dict(zip(food_items,df['Carbohydrates g']))

# Create a dictionary of dietary fiber for all food items
dietary_fiber = dict(zip(food_items,df['Dietary_Fiber g']))

# Create a dictionary of protein for all food items
protein = dict(zip(food_items,df['Protein g']))

# Create a dictionary of Vit_A for all food items
vit_a = dict(zip(food_items,df['Vit_A IU']))

# Create a dictionary of Vit_C for all food items
vit_c = dict(zip(food_items,df['Vit_C IU']))

# Create a dictionary of calcium for all food items
calcium = dict(zip(food_items,df['Calcium mg']))

# Create a dictionary of iron for all food items
iron = dict(zip(food_items,df['Iron mg']))



# Now formulate the problem and solve using linear programming
food_vars = LpVariable.dicts("Food",food_items,lowBound=0,cat='Continuous')
prob += lpSum([costs[i]*food_vars[i] for i in food_items])

# Constraints for total calories intake
prob += lpSum([calories[f] * food_vars[f] for f in food_items]) >= 1500.00
prob += lpSum([calories[f] * food_vars[f] for f in food_items]) <= 2500.00


# Now the constraints for individual categories ##

# Cholesterol
prob += lpSum([cholesterol[f] * food_vars[f] for f in food_items]) >= 30.0, "CholesterolMi
```

```
prob += lpSum([cholesterol[f] * food_vars[f] for f in food_items]) <= 240.0, "CholesterolM

# Fat
prob += lpSum([fat[f] * food_vars[f] for f in food_items]) >= 20.0, "FatMinimum"
prob += lpSum([fat[f] * food_vars[f] for f in food_items]) <= 70.0, "FatMaximum"

# Sodium
prob += lpSum([sodium[f] * food_vars[f] for f in food_items]) >= 800.0, "SodiumMinimum"
prob += lpSum([sodium[f] * food_vars[f] for f in food_items]) <= 2000.0, "SodiumMaximum"

# Carbs
prob += lpSum([carbs[f] * food_vars[f] for f in food_items]) >= 130.0, "CarbsMinimum"
prob += lpSum([carbs[f] * food_vars[f] for f in food_items]) <= 450.0, "CarbsMaximum"

# Dietary Fiber
prob += lpSum([dietary_fiber[f] * food_vars[f] for f in food_items]) >= 125.0, "DietaryFib
prob += lpSum([dietary_fiber[f] * food_vars[f] for f in food_items]) <= 250.0, "DietaryFib

# Protein
prob += lpSum([protein[f] * food_vars[f] for f in food_items]) >= 60.0, "ProteinMinimum"
prob += lpSum([protein[f] * food_vars[f] for f in food_items]) <= 100.0, "ProteinMaximum"

# Vit_A
prob += lpSum([vit_a[f] * food_vars[f] for f in food_items]) >= 1000.0, "VitAMinimum"
prob += lpSum([vit_a[f] * food_vars[f] for f in food_items]) <= 10000.0, "VitAMaximum"

# Vit_C
prob += lpSum([vit_c[f] * food_vars[f] for f in food_items]) >= 400.0, "VitCMinimum"
prob += lpSum([vit_c[f] * food_vars[f] for f in food_items]) <= 5000.0, "VitCMaximum"

# Calcium
prob += lpSum([calcium[f] * food_vars[f] for f in food_items]) >= 700.0, "CalciumMinimum"
prob += lpSum([calcium[f] * food_vars[f] for f in food_items]) <= 1500.0, "CalciumMaximum"

# Iron
prob += lpSum([iron[f] * food_vars[f] for f in food_items]) >= 10.0, "IronMinimum"
prob += lpSum([iron[f] * food_vars[f] for f in food_items]) <= 40.0, "IronMaximum"



# Solve the LPP
prob.solve()
print("Status:", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue>0:
        print(v.name, "=", v.varValue)

obj_fn = value(prob.objective)
print("Total cost: ${}".format(round(obj_fn,2)))
```

```
Status: Optimal
Food_Celery,_Raw = 52.64371
Food_Frozen_Broccoli = 0.25960653
Food_Lettuce,Iceberg,Raw = 63.988506
Food_Oranges = 2.2929389
Food_Poached_Eggs = 0.14184397
```

```
Food_Popcorn,Air_Popped = 13.869322
Total cost: $4.34
```

It can be seen from the above solution that the cost of the balanced diet satisfying the calorie constraint is $4.34 and contains the shown items as per the serving size quantities shown against the items. Here the quantities are real numbers, not integers, which can be problem practically. Hence we can go for integer solution as shown below.

```
## Formulation as Integer programming problem
prob_int = LpProblem("Army_Diet_Problem (Integer Programming)",LpMinimize)
food_int = LpVariable.dicts("Food",food_items,lowBound=0,cat='Integer')
prob_int += lpSum([costs[i]*food_int[i] for i in food_items])

# Now the constraint and everything is encoded as below.
# Constraints for total calories intake
prob_int += lpSum([calories[f] * food_int[f] for f in food_items]) >= 1500.00
prob_int += lpSum([calories[f] * food_int[f] for f in food_items]) <= 2500.00

# Now the constraints for individual categories ##

# Cholesterol
prob_int += lpSum([cholesterol[f] * food_int[f] for f in food_items]) >= 30.0, "Cholestero
prob_int += lpSum([cholesterol[f] * food_int[f] for f in food_items]) <= 240.0, "Cholester

# Fat
prob_int += lpSum([fat[f] * food_int[f] for f in food_items]) >= 20.0, "FatMinimum"
prob_int += lpSum([fat[f] * food_int[f] for f in food_items]) <= 70.0, "FatMaximum"

# Sodium
prob_int += lpSum([sodium[f] * food_int[f] for f in food_items]) >= 800.0, "SodiumMinimum"
prob_int += lpSum([sodium[f] * food_int[f] for f in food_items]) <= 2000.0, "SodiumMaximum

# Carbs
prob_int += lpSum([carbs[f] * food_int[f] for f in food_items]) >= 130.0, "CarbsMinimum"
prob_int += lpSum([carbs[f] * food_int[f] for f in food_items]) <= 450.0, "CarbsMaximum"

# Dietary Fiber
prob_int += lpSum([dietary_fiber[f] * food_int[f] for f in food_items]) >= 125.0, "Dietary
prob_int += lpSum([dietary_fiber[f] * food_int[f] for f in food_items]) <= 250.0, "Dietary

# Protein
prob_int += lpSum([protein[f] * food_int[f] for f in food_items]) >= 60.0, "ProteinMinimum
prob_int += lpSum([protein[f] * food_int[f] for f in food_items]) <= 100.0, "ProteinMaximu

# Vit_A
prob_int += lpSum([vit_a[f] * food_int[f] for f in food_items]) >= 1000.0, "VitAMinimum"
prob_int += lpSum([vit_a[f] * food_int[f] for f in food_items]) <= 10000.0, "VitAMaximum"

# Vit_C
prob_int += lpSum([vit_c[f] * food_int[f] for f in food_items]) >= 400.0, "VitCMinimum"
prob_int += lpSum([vit_c[f] * food_int[f] for f in food_items]) <= 5000.0, "VitCMaximum"
```

```
# Calcium
prob_int += lpSum([calcium[f] * food_int[f] for f in food_items]) >= 700.0, "CalciumMinimu
prob_int += lpSum([calcium[f] * food_int[f] for f in food_items]) <= 1500.0, "CalciumMaxim

# Iron
prob_int += lpSum([iron[f] * food_int[f] for f in food_items]) >= 10.0, "IronMinimum"
prob_int += lpSum([iron[f] * food_int[f] for f in food_items]) <= 40.0, "IronMaximum"

# Solve the Integer Programming
prob_int.solve()
print("Status:", LpStatus[prob_int.status])
for v in prob_int.variables():
    if v.varValue>0:
        print(v.name, "=", v.varValue)

obj_fn = value(prob_int.objective)
print("The total cost of integer solution is: ${}".format(round(obj_fn,2)))
```

```
/usr/local/lib/python3.7/dist-packages/pulp/pulp.py:1352: UserWarning: Spaces are not
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
Status: Optimal
Food_Celery,_Raw = 41.0
Food_Kiwifruit,Raw,Fresh = 1.0
Food_Lettuce,Iceberg,Raw = 91.0
Food_Oranges = 2.0
Food_Poached_Eggs = 1.0
Food_Popcorn,Air_Popped = 14.0
The total cost of integer solution is: $4.89
```

From the above integer solution, we see that the cost has increased and it is $4.89. But in this solution, the servings of each item in the diet are given integer values which is more practical solution.

Problem: 15.2, 2(a): In this case, we add following constraints to real valued LPP problem definition as follows and then solve. Note that now the solution prints the "Chosen" variables also to indicate that a particular food item is chosen.

```
# Food item is chosen or not chosen
food_chosen = LpVariable.dicts("Chosen",food_items,0,1,cat='Integer')
# COnstraint specifying that minimum 1/10 serving must be chosen
for f in food_items:
    prob += food_vars[f]>= food_chosen[f]*0.1
    prob += food_vars[f]<= food_chosen[f]*1e5

prob.solve()
print("Status:", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue>0:
        print(v.name, "=", v.varValue)
```

```
obj_fn = value(prob.objective)
print("The total cost of this solution is: ${}".format(round(obj_fn,2)))
```

```
Status: Optimal
Chosen_Celery,_Raw = 1.0
Chosen_Frozen_Broccoli = 1.0
Chosen_Lettuce,Iceberg,Raw = 1.0
Chosen_Oranges = 1.0
Chosen_Poached_Eggs = 1.0
Chosen_Popcorn,Air_Popped = 1.0
Food_Celery,_Raw = 52.64371
Food_Frozen_Broccoli = 0.25960653
Food_Lettuce,Iceberg,Raw = 63.988506
Food_Oranges = 2.2929389
Food_Poached_Eggs = 0.14184397
Food_Popcorn,Air_Popped = 13.869322
The total cost of this solution is: $4.34
```

Problem: 15.2, 2(b): Adding one more constraint for mutual exclusion of celery and frozen broccoli.

```
prob += food_chosen['Frozen Broccoli']+food_chosen['Celery, Raw']<=1

prob.solve()
print("Status:", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue>0:
        print(v.name, "=", v.varValue)

obj_fn = value(prob.objective)
print("The total cost of solution is: ${}".format(round(obj_fn,2)))
```

```
Status: Optimal
Chosen_Celery,_Raw = 1.0
Chosen_Lettuce,Iceberg,Raw = 1.0
Chosen_Oranges = 1.0
Chosen_Peanut_Butter = 1.0
Chosen_Poached_Eggs = 1.0
Chosen_Popcorn,Air_Popped = 1.0
Food_Celery,_Raw = 43.154119
Food_Lettuce,Iceberg,Raw = 80.919121
Food_Oranges = 3.0765161
Food_Peanut_Butter = 2.0464575
Food_Poached_Eggs = 0.14184397
Food_Popcorn,Air_Popped = 13.181772
The total cost of solution is: $4.49
```

Problem: 15.2, 2(c): New constraint for at least 3 types of meat/poultry/fish/eggs. For this solution, we have considered the shown options as candidates for meat/poultry/fish/eggs. There can be other combinations also.

```
prob += food_chosen['Roasted Chicken']+food_chosen['Poached Eggs']+food_chosen['Scrambled
```

```
prob.solve()
print("Status:", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue>0:
        print(v.name, "=", v.varValue)

obj_fn = value(prob.objective)
print("The total cost of this balanced diet is: ${}".format(round(obj_fn,2)))
```

```
Status: Optimal
Chosen_Bologna,Turkey = 1.0
Chosen_Celery,_Raw = 1.0
Chosen_Lettuce,Iceberg,Raw = 1.0
Chosen_Oranges = 1.0
Chosen_Peanut_Butter = 1.0
Chosen_Poached_Eggs = 1.0
Chosen_Popcorn,Air_Popped = 1.0
Chosen_Scrambled_Eggs = 1.0
Food_Bologna,Turkey = 0.1
Food_Celery,_Raw = 42.423026
Food_Lettuce,Iceberg,Raw = 82.673927
Food_Oranges = 3.0856009
Food_Peanut_Butter = 1.9590978
Food_Poached_Eggs = 0.1
Food_Popcorn,Air_Popped = 13.214473
Food_Scrambled_Eggs = 0.1
The total cost of this balanced diet is: $4.51
```

✓ 0s completed at 10:40 PM