Homework 12: Optimization

In [1]:
```
# Dependencies
import pulp
import pandas as pd
import numpy as np
```

Question 1

Problem: Formulate an optimization model (linear program) to find cheapest diet satisfying the max and min daily nutrition constraints; solve using Pulp.

Read Data

First, we begin by just looking at the data. Even looking at it in Excel, we see that it is neatly organized into tabular format. Most of the variables are numerical, except 'Serving Size' being strings.

Second, we can see the bottom information about the constraints.

In [2]:
```
# Original dataframes
diet = pd.read_excel('Data/diet.xls', nrows=64)
dietL = pd.read_excel('Data/diet_large.xls', skiprows=1)

# Extract the bottom constraints
diet_constraints = pd.read_excel('Data/diet.xls',
                                 skiprows=66, header=None
                                 ).iloc[:,2:]
diet_constraints.columns = diet.columns[2:]
```

In [3]:
```
diet_constraints
```

Out[3]:

	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Dietary_Fiber g	Protein g	Vit_A IU
0	Minimum daily intake	1500	30	20	800	130	125	60	1000
1	Maximum daily intake	2500	240	70	2000	450	250	100	10000

In [4]:
```
1 diet.head(3)
```

Out[4]:

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Dietary
0	Frozen Broccoli	0.16	10 Oz Pkg	73.8	0.0	0.8	68.2	13.6	
1	Carrots,Raw	0.07	1/2 Cup Shredded	23.7	0.0	0.1	19.2	5.6	
2	Celery, Raw	0.04	1 Stalk	6.4	0.0	0.1	34.8	1.5	

If you haven't heard of Pandas already, it seems to be the most common data manipulation library in Python for data science/ ML work at a basic level.

Here, we see that the large_diet has a line skipped in the Excel file so we can just skip a row to neatly organize it into a viewable format. Other than this, the data is already cleaned so no further operations are necessary.

Optimizing with Pulp

First, list comprehension in Python is an extremely useful tool. Essentially, lists are arrays. We can create lists iteratively using a for loop in just one line of code.

Since there are so many different food variables each with its own set of nutritional values, prices, and min/max constraints, it would make sense to loop through these variables.

Furthermore, looking at this as just a large matrix multiplication as seen in the office hour's Pulp Example (https://pythonhosted.org/PuLP/CaseStudies/a_blending_problem.html), each row of equations is just a dot product (or sum of multiplications) between the nutritional value and the food value names.
(e.g. 1.03 * Food1 + 2 * Food2)

We can put all of this together in a for loop to define the min and max constraints.

In [5]:
```
1 # Problem: Minimize the Cost
2 prob = pulp.LpProblem("Army Diet Optimization", pulp.LpMinimize)
```

In [6]:
```
1 food = diet['Foods']
2 varList = pulp.LpVariable.dicts("Foods", food, lowBound=0)
3
4 # Binary variable
5 binList = pulp.LpVariable.dicts("Binary",food, lowBound=0, upBound=1, c
```

In [7]:
```
1 # Objective Function:
2 #     minimizing the (price/servings)*foods
3 price = list(diet['Price/ Serving'])
4 prob += pulp.lpSum([price[i]*varList[food[i]] for i in range(len(varLis
```

In [8]:
```
# Constraints
#     nutrient*food > min
#     nutrients*food < max

# List of nutritients
col = diet_constraints.columns[1:]

# Set min/max constraints for each food type
for c in col:
    prob += pulp.lpSum([diet[c][i]*varList[food[i]] for i in range(len(
    prob += pulp.lpSum([diet[c][i]*varList[food[i]] for i in range(len(
```

In [9]:
```
# Save problem in DietModel.lp
# Solve problem
prob.solve()
```

Out[9]: 1

In [10]:
```
# Found optimal solution
print("Status:", pulp.LpStatus[prob.status])
```

```
Status: Optimal
```

In [11]:
```
# Print the solution
for v in prob.variables():
    if v.varValue != 0:
        print(v.name, "=", v.varValue)
```

```
Foods_Celery,_Raw = 52.64371
Foods_Frozen_Broccoli = 0.25960653
Foods_Lettuce,Iceberg,Raw = 63.988506
Foods_Oranges = 2.2929389
Foods_Poached_Eggs = 0.14184397
Foods_Popcorn,Air_Popped = 13.869322
```

Constructing Optimal Solutions

Using panda's awesome ability to manipulte DataFrames, we can construct several optimal solutions and export them into Excel or save them for convenience.

First, we extract the variable names and their optimal servings amount solved for in the 'Optimizing with Pulp' section. It seems that pulp replaces spaces with an underscore, so we can revert the change during our extraction process.

Second, we save these values in a Pandas DataFrame and export to Excel.

Third, we join the original 'diet.xls' table with the optimal solutions, adding another column with the 'Optimal Amount'.

In [12]:
```
# Dataframe of optimal values
var = prob.variables()
df_optimal = pd.DataFrame(
    {'Foods':[v.name.replace("_", " ").replace("Foods ","") for v in va
     'Optimal Servings 1': [v.varValue for v in var]},
)
```

In [13]:
```
# Join table with optimal solution and export to Excel
diet_optimal = diet.merge(df_optimal, on='Foods')
```

In [14]:
```
# DataFrame containing just the necessary values
df1 = diet_optimal.loc[diet_optimal['Optimal Servings 1'] > 0 ][['Foods
df1
```

Out[14]:

	Foods	Price/ Serving	Optimal Servings 1
0	Frozen Broccoli	0.16	0.259607
2	Celery, Raw	0.04	52.643710
4	Lettuce,Iceberg,Raw	0.02	63.988506
13	Oranges	0.15	2.292939
25	Poached Eggs	0.08	0.141844

In [15]:
```
# Solve for cost
np.sum(df1['Price/ Serving'] * df1['Optimal Servings 1'])
```

Out[15]:
```
3.7823439173999995
```

Total cost per day per one member of the army is $3.78

Question 2

Problem: solve optimization problem as before but with additional logical constraints.
We will add the constraints one at a time - combining these, we will get a new set of food with all of the constraints together.

a.) Minimum of 0.1 servings if used at all

The value must be greater than 0.1 if used at all.
The second part ties in to part b. of the solution. By stating that the food value must be less than the multiple of the binary value, if the binary value is 0 then the food value is also 0. However, this limits the amount that the food value can be if the binary is 1. Therefore, we put an extremely large coefficient to the binary expression so that we can have at least that many servings.

It is importnat to note that we are losing out on any potential solutions in which the amount of servings is greater than the coefficient placed with the binary value. To prevent such a case, the binary value is large enough beyond a reasonable solution to the optimization problem.

In [16]:
```
for i in food:
    # Sets minimum to 0.1
    prob += varList[i] >= 0.1*binList[i]

    # Ties food value with binary value
    prob += varList[i] <= 9001*binList[i]
```

b.) Frozen Broccoli and Raw Celery are mutually exclusive

Simple enough, we set the value 1 so that at least one of the two must be used but not more than the one.

In [17]:
```
# Either one or the other
prob += binList['Frozen Broccoli'] + binList['Celery, Raw'] == 1
```

c.) At least 3 proteins

This will take some manual sorting, since there is no categorical variable indicating whether or not a food is a meat or egg. We can sort the values to make this easier. Some of the foods are a bit iffy, but I included anything with a mention of meat or eggs in it.

Beyond that, the constraints are just as in part b. As long as the number of binary values equals a certain number, that many foods will be included in the model. Note that the original constraint is still in place, and any food included will at least have 0.1 servings.

In [18]:
```
np.sort(food)
```

Out[18]:
```
array(['2% Lowfat Milk', '3.3% Fat,Whole Milk', 'Apple Pie',
       'Apple,Raw,W/Skin', 'Bagels', 'Banana', 'Beanbacn Soup,W/Watr',
       'Bologna,Turkey', 'Butter,Regular', "Cap'N Crunch", 'Carrots,Raw',
       'Celery, Raw', 'Cheddar Cheese', 'Cheerios', 'Chicknoodl Soup',
       'Chocolate Chip Cookies', "Corn Flks, Kellogg'S", 'Couscous',
       'Crm Mshrm Soup,W/Mlk', 'Frankfurter, Beef', 'Frozen Broccoli',
       'Frozen Corn', 'Grapes', 'Ham,Sliced,Extralean',
       'Hamburger W/Toppings', 'Hotdog, Plain', 'Kielbasa,Prk',
       'Kiwifruit,Raw,Fresh', 'Lettuce,Iceberg,Raw', 'Macaroni,Ckd',
       'Malt-O-Meal,Choc', 'New E Clamchwd,W/Mlk', 'Neweng Clamchwd',
       'Oatmeal', 'Oatmeal Cookies', 'Oranges', 'Peanut Butter',
       'Peppers, Sweet, Raw', 'Pizza W/Pepperoni', 'Poached Eggs',
       'Popcorn,Air-Popped', 'Pork', 'Potato Chips,Bbqflvr',
       'Potatoes, Baked', 'Pretzels', "Raisin Brn, Kellg'S",
       'Rice Krispies', 'Roasted Chicken', 'Sardines in Oil',
       'Scrambled Eggs', 'Skim Milk', 'Spaghetti W/ Sauce', 'Special K',
       'Splt Pea&Hamsoup', 'Taco', 'Tofu', 'Tomato Soup',
       'Tomato,Red,Ripe,Raw', 'Tortilla Chip', 'Vegetbeef Soup',
       'Wheat Bread', 'White Bread', 'White Rice', 'White Tuna in Wate
r'],
      dtype=object)
```

In [19]:
```
# List of necessary proteins
proteinList = [
       'Bologna,Turkey', 'Frankfurter, Beef','Ham,Sliced,Extralean',
       'Hamburger W/Toppings', 'Hotdog, Plain', 'Kielbasa,Prk',
       'Pizza W/Pepperoni', 'Poached Eggs',
       'Pork', 'Roasted Chicken', 'Sardines in Oil',
       'Scrambled Eggs',
       'Splt Pea&Hamsoup', 'Vegetbeef Soup',
       'White Tuna in Water']
```

In [20]:
```
# Build constraints of at least 3
prob += pulp.lpSum([binList[p] for p in proteinList]) >= 3
```

In [21]:
```
prob.solve()
```

Out[21]: 1

In [22]:
```
# Dataframe of optimal values
var = prob.variables()
df_optimal2 = pd.DataFrame(
    {'Foods':[v.name.replace("_", " ").replace("Foods ","") for v in va
     'Optimal Servings 2': [v.varValue for v in var]},
)

# Join table with optimal solution and export to Excel
diet_optimal = diet_optimal.merge(df_optimal2, on='Foods')
```

In [23]:
```
# DataFrame containing just the necessary values
df2 = diet_optimal.loc[diet_optimal['Optimal Servings 2'] >0][['Foods',
```

In [24]:
```
df2
```

Out[24]:

	Foods	Price/ Serving	Optimal Servings 2
2	Celery, Raw	0.04	42.399358
4	Lettuce,Iceberg,Raw	0.02	82.802586
13	Oranges	0.15	3.077184
25	Poached Eggs	0.08	0.100000
26	Scrambled Eggs	0.11	0.100000
30	Kielbasa,Prk	0.15	0.100000
43	Peanut Butter	0.07	1.942972

Cost

In [25]:
```
# Solve for cost
np.sum(df2['Price/ Serving'] * df2['Optimal Servings 2'])
```

Out[25]: 3.9836116670000004

Note, the cost here is a little higher, at $3.98 per day for each army member. Of couse, with more constraints, this is to be expected. As we place more restrictions on the optimization, there is less room for it to move and improve.

Similarly, the same occurs for linear regression when we optimize for the coefficients. As we add a regularization term to the Loss function, we say give the variables less wriggle room to reach its minimum values.