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.]

In [1]:
```
#!pip install PuLP
#!pip install xlrd
import pandas as pd
import pulp
from pulp import *
```

EDA

In [2]:
```
excel_file = r"C:\Users\Clair\OneDrive\Documents\GitHub\omsa\ISYE 6501\Homework 11\

# read by default 1st sheet of an excel file
diet_data = pd.read_excel(excel_file)
```

```
# table preview
display(diet_data)
```

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohy
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	
...	...	...	...	...	...	...	...	
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	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
65	NaN	NaN	Minimum daily intake	1500.0	30.0	20.0	800.0	
66	NaN	NaN	Maximum daily intake	2500.0	240.0	70.0	2000.0	

67 rows × 14 columns

In [3]:
```
diet_data.describe()
```

Out[3]:

	Price/ Serving	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Dietary
count	64.000000	66.000000	66.000000	66.000000	66.000000	66.000000	66.0
mean	0.327188	190.918182	21.615152	6.392424	364.486364	24.727273	7.1
std	0.254536	354.515790	49.660770	12.547158	528.629735	57.824484	34.0
min	0.020000	2.600000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.145000	73.975000	0.000000	0.500000	17.175000	5.000000	0.0
50%	0.270000	110.650000	0.000000	2.900000	144.850000	15.250000	0.8
75%	0.460000	168.950000	19.425000	7.075000	389.025000	22.450000	2.0
max	0.990000	2500.000000	240.000000	72.200000	2000.000000	450.000000	250.0

In [4]:
```
# I see NaNs in the table preview
diet_data.isnull().values.any()
```

Out[4]:
```
True
```

In [5]:
```
# how many null rows?
diet_data.isnull().sum()
```

Out[5]:
```
Foods              3
Price/ Serving     3
Serving Size       1
Calories           1
Cholesterol mg     1
Total_Fat g        1
Sodium mg          1
Carbohydrates g    1
Dietary_Fiber g    1
Protein g          1
Vit_A IU           1
Vit_C IU           1
Calcium mg         1
Iron mg            1
dtype: int64
```

In [6]:
```
# what are those rows, more importantly i want to know the indices
diet_data[diet_data.isnull().any(axis=1)]
```

Out[6]:

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Di
64	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
65	NaN	NaN	Minimum daily intake	1500.0	30.0	20.0	800.0	130.0	
66	NaN	NaN	Maximum daily intake	2500.0	240.0	70.0	2000.0	450.0	

In [7]:
```
# limit rows to just legit food rows
diet_data_clean = diet_data[0:64]
# making sure the null value columns are truly eliminated
diet_data_clean.tail()
```

Out[7]:

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydı
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	

Min and max serving

1. Orgnaize data

In [8]:
```
# selecting food rows (the rows that are not null)
food_data = diet_data_clean
nutrient_constraints = food_data.iloc[:, 3:].reset_index(drop=True) # every row of
display(food_data.head(2)) # every row of the diet data clean --> diet_data_clean =
display(nutrient_constraints.head(2)) # 4th column and on starting with calories
```

	Foods	Price/ Serving	Serving Size	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g
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

	Calories	Cholesterol mg	Total_Fat g	Sodium mg	Carbohydrates g	Dietary_Fiber g	Protein g	Vit_A IU
0	73.8	0.0	0.8	68.2	13.6	8.5	8.0	5867.4
1	23.7	0.0	0.1	19.2	5.6	1.6	0.6	15471.0

In [9]:
```
# extract costs and nutrient values
food_costs = food_data['Price/ Serving'].tolist()
calories = food_data['Calories'].tolist()
cholestrols = food_data['Cholesterol mg'].tolist()
fats = food_data['Total_Fat g'].tolist()
sodium = food_data['Sodium mg'].tolist()
carbohydrates = food_data['Carbohydrates g'].tolist()
fibers = food_data['Dietary_Fiber g'].tolist()
vitamin_a = food_data['Vit_A IU'].tolist()
vitamin_c = food_data['Vit_C IU'].tolist()
calcium = food_data['Calcium mg'].tolist()
iron = food_data['Iron mg'].tolist()

print(food_costs)
```

```
[0.16, 0.07, 0.04, 0.18, 0.02, 0.53, 0.06, 0.31, 0.84, 0.78, 0.27, 0.24, 0.15, 0.32,
0.49, 0.15, 0.16, 0.05, 0.06, 0.09, 0.16, 0.03, 0.05, 0.25, 0.16, 0.23, 0.13, 0.08,
0.11, 0.15, 0.27, 0.33, 0.15, 0.31, 0.28, 0.28, 0.34, 0.32, 0.38, 0.82, 0.52, 0.44,
0.59, 0.83, 0.31, 0.39, 0.08, 0.17, 0.07, 0.81, 0.45, 0.69, 0.04, 0.22, 0.12, 0.19,
0.39, 0.67, 0.71, 0.75, 0.39, 0.99, 0.65, 0.67]
```

2. Pulp object

In [10]:
```
import pulp
prob = pulp.LpProblem("diet_optimization", sense=pulp.LpMinimize)
prob
```

Out[10]:
```
diet_optimization:
MINIMIZE
None
VARIABLES
```

2.1 Decision variables

```
    # food_vars = [pulp.LpVariable(f"x_{food_data['Foods'][i]}",
    lowBound=0, cat='LpContinuous') for i in range(len(food_data))]
    # food_vars
```

In [11]:
```
x = [0]*len(food_data)
# vector of variables containing each food var
for i, row in food_data.iterrows():
    x[i] = pulp.LpVariable(row['Foods'], 0, None, pulp.LpContinuous)

x
```

Out[11]: [Frozen_Broccoli,
Carrots,Raw,
Celery,_Raw,
Frozen_Corn,
Lettuce,Iceberg,Raw,
Peppers,_Sweet,_Raw,
Potatoes,_Baked,
Tofu,
Roasted_Chicken,
Spaghetti_W__Sauce,
Tomato,Red,Ripe,Raw,
Apple,Raw,W_Skin,
Banana,
Grapes,
Kiwifruit,Raw,Fresh,
Oranges,
Bagels,
Wheat_Bread,
White_Bread,
Oatmeal_Cookies,
Apple_Pie,
Chocolate_Chip_Cookies,
Butter,Regular,
Cheddar_Cheese,
3.3%_Fat,Whole_Milk,
2%_Lowfat_Milk,
Skim_Milk,
Poached_Eggs,
Scrambled_Eggs,
Bologna,Turkey,
Frankfurter,_Beef,
Ham,Sliced,Extralean,
Kielbasa,Prk,
Cap'N_Crunch,
Cheerios,
Corn_Flks,_Kellogg'S,
Raisin_Brn,_Kellg'S,
Rice_Krispies,
Special_K,
Oatmeal,
Malt_O_Meal,Choc,
Pizza_W_Pepperoni,
Taco,
Hamburger_W_Toppings,
Hotdog,_Plain,
Couscous,
White_Rice,
Macaroni,Ckd,
Peanut_Butter,
Pork,
Sardines_in_Oil,
White_Tuna_in_Water,
Popcorn,Air_Popped,
Potato_Chips,Bbqflvr,
Pretzels,
Tortilla_Chip,

```
Chicknoodl_Soup,
Splt_Pea&Hamsoup,
Vegetbeef_Soup,
Neweng_Clamchwd,
Tomato_Soup,
New_E_Clamchwd,W_Mlk,
Crm_Mshrm_Soup,W_Mlk,
Beanbacn_Soup,W_Watr]
```

2.2 Add objective function

minimizing total food cost

```
# prob = pulp.lpSum([food_costs[i] * food_vars[i] for i in
range(len(food_data))]), 'Total Cost'
# prob
```

In [12]:
```
# objective function by multiplying each variable by its associated cost and summin
prob += sum(food_data['Price/ Serving']*x)
prob
```

Out[12]:
```
diet_optimization:
MINIMIZE
0.23*2%_Lowfat_Milk + 0.16*3.3%_Fat,Whole_Milk + 0.24*Apple,Raw,W_Skin + 0.16*Appl
e_Pie + 0.16*Bagels + 0.15*Banana + 0.67*Beanbacn_Soup,W_Watr + 0.15*Bologna,Turke
y + 0.05*Butter,Regular + 0.31*Cap'N_Crunch + 0.07*Carrots,Raw + 0.04*Celery,_Raw
+ 0.25*Cheddar_Cheese + 0.28*Cheerios + 0.39*Chicknoodl_Soup + 0.03*Chocolate_Chip
_Cookies + 0.28*Corn_Flks,_Kellogg'S + 0.39*Couscous + 0.65*Crm_Mshrm_Soup,W_Mlk +
0.27*Frankfurter,_Beef + 0.16*Frozen_Broccoli + 0.18*Frozen_Corn + 0.32*Grapes +
0.33*Ham,Sliced,Extralean + 0.83*Hamburger_W_Toppings + 0.31*Hotdog,_Plain + 0.15*
Kielbasa,Prk + 0.49*Kiwifruit,Raw,Fresh + 0.02*Lettuce,Iceberg,Raw + 0.17*Macaron
i,Ckd + 0.52*Malt_O_Meal,Choc + 0.99*New_E_Clamchwd,W_Mlk + 0.75*Neweng_Clamchwd +
0.82*Oatmeal + 0.09*Oatmeal_Cookies + 0.15*Oranges + 0.07*Peanut_Butter + 0.53*Pep
pers,_Sweet,_Raw + 0.44*Pizza_W_Pepperoni + 0.08*Poached_Eggs + 0.04*Popcorn,Air_P
opped + 0.81*Pork + 0.22*Potato_Chips,Bbqflvr + 0.06*Potatoes,_Baked + 0.12*Pretze
ls + 0.34*Raisin_Brn,_Kellg'S + 0.32*Rice_Krispies + 0.84*Roasted_Chicken + 0.45*S
ardines_in_Oil + 0.11*Scrambled_Eggs + 0.13*Skim_Milk + 0.78*Spaghetti_W__Sauce +
0.38*Special_K + 0.67*Splt_Pea&Hamsoup + 0.59*Taco + 0.31*Tofu + 0.27*Tomato,Red,R
ipe,Raw + 0.39*Tomato_Soup + 0.19*Tortilla_Chip + 0.71*Vegetbeef_Soup + 0.05*Wheat
_Bread + 0.06*White_Bread + 0.08*White_Rice + 0.69*White_Tuna_in_Water + 0.0
VARIABLES
2%_Lowfat_Milk Continuous
3.3%_Fat,Whole_Milk Continuous
Apple,Raw,W_Skin Continuous
Apple_Pie Continuous
Bagels Continuous
Banana Continuous
Beanbacn_Soup,W_Watr Continuous
Bologna,Turkey Continuous
Butter,Regular Continuous
Cap'N_Crunch Continuous
Carrots,Raw Continuous
Celery,_Raw Continuous
Cheddar_Cheese Continuous
Cheerios Continuous
Chicknoodl_Soup Continuous
Chocolate_Chip_Cookies Continuous
Corn_Flks,_Kellogg'S Continuous
Couscous Continuous
Crm_Mshrm_Soup,W_Mlk Continuous
Frankfurter,_Beef Continuous
Frozen_Broccoli Continuous
Frozen_Corn Continuous
Grapes Continuous
Ham,Sliced,Extralean Continuous
Hamburger_W_Toppings Continuous
Hotdog,_Plain Continuous
Kielbasa,Prk Continuous
Kiwifruit,Raw,Fresh Continuous
Lettuce,Iceberg,Raw Continuous
Macaroni,Ckd Continuous
Malt_O_Meal,Choc Continuous
New_E_Clamchwd,W_Mlk Continuous
Neweng_Clamchwd Continuous
Oatmeal Continuous
Oatmeal_Cookies Continuous
Oranges Continuous
```

```
Peanut_Butter Continuous
Peppers,_Sweet,_Raw Continuous
Pizza_W_Pepperoni Continuous
Poached_Eggs Continuous
Popcorn,Air_Popped Continuous
Pork Continuous
Potato_Chips,Bbqflvr Continuous
Potatoes,_Baked Continuous
Pretzels Continuous
Raisin_Brn,_Kellg'S Continuous
Rice_Krispies Continuous
Roasted_Chicken Continuous
Sardines_in_Oil Continuous
Scrambled_Eggs Continuous
Skim_Milk Continuous
Spaghetti_W__Sauce Continuous
Special_K Continuous
Splt_Pea&Hamsoup Continuous
Taco Continuous
Tofu Continuous
Tomato,Red,Ripe,Raw Continuous
Tomato_Soup Continuous
Tortilla_Chip Continuous
Vegetbeef_Soup Continuous
Wheat_Bread Continuous
White_Bread Continuous
White_Rice Continuous
White_Tuna_in_Water Continuous
```

2.3 Add constraints

(min and max nutrients)

```
#
# prob += pulp.lpSum([calories[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,0],
"min_calories"
# prob += pulp.lpSum([calories[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,0],
"max_calories"

# # Cholesterols
# prob += pulp.lpSum([cholestrols[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,1],
"min_cholestrols"
# prob += pulp.lpSum([cholestrols[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,1],
"max_cholestrols"

# # Fats
# prob += pulp.lpSum([fats[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,2],
"min_fats"
```

```
# prob += pulp.lpSum([fats[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,2],
"max_fats"

# # Sodium
# prob += pulp.lpSum([sodium[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,3],
"min_sodium"
# prob += pulp.lpSum([sodium[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,3],
"max_sodium"

# # Carbohydrates
# prob += pulp.lpSum([carbohydrates[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,4],
"min_carbohydrates"
# prob += pulp.lpSum([carbohydrates[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,4],
"max_carbohydrates"

# # Fibers
# prob += pulp.lpSum([fibers[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,5],
"min_fibers"
# prob += pulp.lpSum([fibers[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,5],
"max_fibers"

# # Vitamin A
# prob += pulp.lpSum([vitamin_a[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,6],
"min_vitamin_a"
# prob += pulp.lpSum([vitamin_a[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,6],
"max_vitamin_a"

# # Vitamin C
# prob += pulp.lpSum([vitamin_c[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,7],
"min_vitamin_c"
# prob += pulp.lpSum([vitamin_c[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,7],
"max_vitamin_c"

# # Calcium
# prob += pulp.lpSum([calcium[i] * food_vars[i] for i in
range(len(food_data))]) >= nutrient_constraints.iloc[0,8],
"min_calcium"
# prob += pulp.lpSum([calcium[i] * food_vars[i] for i in
range(len(food_data))]) <= nutrient_constraints.iloc[1,8],
"max_calcium"
```

```
    # # Iron
    # prob += pulp.lpSum([iron[i] * food_vars[i] for i in
    range(len(food_data))]) >= nutrient_constraints.iloc[0,9],
    "min_iron"
    # prob += pulp.lpSum([iron[i] * food_vars[i] for i in
    range(len(food_data))]) <= nutrient_constraints.iloc[1,9],
    "max_iron"
```

In [13]:
```
# before importing the data, looking at the excel sheet, i can see Lettuce,Iceberg,

# lowest and highest calories
min_calories_index = food_data['Calories'].idxmin()
max_calories_index = food_data['Calories'].idxmax()

# what row are these foods in
print("Index of minimum Calories:", min_calories_index)
print("Index of maximum Calories:", max_calories_index)

# lets see the data for the foods im anticipating
min_calories_row = food_data.loc[min_calories_index] # should be Lettuce,Iceberg,Ra
max_calories_row = food_data.loc[max_calories_index] # should be Pork

print("Row with minimum Calories:\n", min_calories_row)
print("Row with maximum Calories:\n", max_calories_row)
```

```
Index of minimum Calories: 4
Index of maximum Calories: 49
Row with minimum Calories:
 Foods              Lettuce,Iceberg,Raw
Price/ Serving                     0.02
Serving Size                    1 Leaf
Calories                           2.6
Cholesterol mg                     0.0
Total_Fat g                        0.0
Sodium mg                          1.8
Carbohydrates g                    0.4
Dietary_Fiber g                    0.3
Protein g                          0.2
Vit_A IU                          66.0
Vit_C IU                           0.8
Calcium mg                         3.8
Iron mg                            0.1
Name: 4, dtype: object
Row with maximum Calories:
 Foods                Pork
Price/ Serving       0.81
Serving Size         4 Oz
Calories            710.8
Cholesterol mg      105.1
Total_Fat g          72.2
Sodium mg            38.4
Carbohydrates g       0.0
Dietary_Fiber g       0.0
Protein g            13.8
Vit_A IU             14.7
Vit_C IU              0.0
Calcium mg           59.9
Iron mg               0.4
Name: 49, dtype: object
```

In [14]:
```
food_columns = list(food_data.columns)
print(food_columns)
mins = min_calories_row
maxes = max_calories_row
print(mins)
print(maxes)
for nutrient in food_columns[3:]:
    prob += sum(x*food_data[nutrient]) >= mins[nutrient]
    prob += sum(x*food_data[nutrient]) <= maxes[nutrient]
```

```
['Foods', 'Price/ Serving', 'Serving Size', 'Calories', 'Cholesterol mg', 'Total_Fat
g', 'Sodium mg', 'Carbohydrates g', 'Dietary_Fiber g', 'Protein g', 'Vit_A IU', 'Vit
_C IU', 'Calcium mg', 'Iron mg']
Foods              Lettuce,Iceberg,Raw
Price/ Serving                   0.02
Serving Size                  1 Leaf
Calories                         2.6
Cholesterol mg                   0.0
Total_Fat g                      0.0
Sodium mg                        1.8
Carbohydrates g                  0.4
Dietary_Fiber g                  0.3
Protein g                        0.2
Vit_A IU                        66.0
Vit_C IU                         0.8
Calcium mg                       3.8
Iron mg                          0.1
Name: 4, dtype: object
Foods               Pork
Price/ Serving      0.81
Serving Size        4 Oz
Calories           710.8
Cholesterol mg     105.1
Total_Fat g         72.2
Sodium mg           38.4
Carbohydrates g      0.0
Dietary_Fiber g      0.0
Protein g           13.8
Vit_A IU            14.7
Vit_C IU             0.0
Calcium mg          59.9
Iron mg              0.4
Name: 49, dtype: object
```

In [15]:
```
prob.writeLP("simple_diet.lp")
```

Out[15]:
```
[2%_Lowfat_Milk,
 3.3%_Fat,Whole_Milk,
 Apple,Raw,W_Skin,
 Apple_Pie,
 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,Red,Ripe,Raw,
 Tomato_Soup,
 Tortilla_Chip,
 Vegetbeef_Soup,
 Wheat_Bread,
 White_Bread,
 White_Rice,
 White_Tuna_in_Water]
```

2.4 Solve prob

In [16]:
```
prob = pulp.LpProblem("simple_diet", pulp.LpMinimize)
prob.solve()
print("Status:", pulp.LpStatus[prob.status]) # should be optimal
```

```
Status: Optimal
```

In [17]:
```
# Print out any food with a solved value greater than 0
for v in prob.variables():
    if v.varValue is not None and v.varValue > 0:
        print(v.name, "=", v.varValue)
```

In [18]:
```
import pulp

# Define the problem
prob = pulp.LpProblem("simple_diet", pulp.LpMinimize)

# Example: Get food costs from your dataset
food_costs = food_data['Price/ Serving'].tolist()
print(food_costs)

# Create decision variables for each food item
food_vars = [pulp.LpVariable(f"x_{food_data['Foods'][i]}", lowBound=0, cat='LpConti
print(food_vars)

# Define the objective: Minimize the total cost (sum of cost * quantity of food)
prob += pulp.lpSum([food_costs[i] * food_vars[i] for i in range(len(food_costs))]),

# Solve the problem
prob.solve()

# Check if the solution is optimal
if pulp.LpStatus[prob.status] == "Optimal":
    print("\nTotal Cost of Ingredients per day = $", round(pulp.value(prob.objectiv
else:
    print("No optimal solution. Status:", pulp.LpStatus[prob.status])
```

```
[0.16, 0.07, 0.04, 0.18, 0.02, 0.53, 0.06, 0.31, 0.84, 0.78, 0.27, 0.24, 0.15, 0.32,
0.49, 0.15, 0.16, 0.05, 0.06, 0.09, 0.16, 0.03, 0.05, 0.25, 0.16, 0.23, 0.13, 0.08,
0.11, 0.15, 0.27, 0.33, 0.15, 0.31, 0.28, 0.28, 0.34, 0.32, 0.38, 0.82, 0.52, 0.44,
0.59, 0.83, 0.31, 0.39, 0.08, 0.17, 0.07, 0.81, 0.45, 0.69, 0.04, 0.22, 0.12, 0.19,
0.39, 0.67, 0.71, 0.75, 0.39, 0.99, 0.65, 0.67]
[x_Frozen_Broccoli, x_Carrots,Raw, x_Celery,_Raw, x_Frozen_Corn, x_Lettuce,Iceberg,R
aw, x_Peppers,_Sweet,_Raw, x_Potatoes,_Baked, x_Tofu, x_Roasted_Chicken, x_Spaghetti
_W__Sauce, x_Tomato,Red,Ripe,Raw, x_Apple,Raw,W_Skin, x_Banana, x_Grapes, x_Kiwifrui
t,Raw,Fresh, x_Oranges, x_Bagels, x_Wheat_Bread, x_White_Bread, x_Oatmeal_Cookies, x
_Apple_Pie, x_Chocolate_Chip_Cookies, x_Butter,Regular, x_Cheddar_Cheese, x_3.3%_Fa
t,Whole_Milk, x_2%_Lowfat_Milk, x_Skim_Milk, x_Poached_Eggs, x_Scrambled_Eggs, x_Bol
ogna,Turkey, x_Frankfurter,_Beef, x_Ham,Sliced,Extralean, x_Kielbasa,Prk, x_Cap'N_Cr
unch, x_Cheerios, x_Corn_Flks,_Kellogg'S, x_Raisin_Brn,_Kellg'S, x_Rice_Krispies, x_
Special_K, x_Oatmeal, x_Malt_O_Meal,Choc, x_Pizza_W_Pepperoni, x_Taco, x_Hamburger_W
_Toppings, x_Hotdog,_Plain, x_Couscous, x_White_Rice, x_Macaroni,Ckd, x_Peanut_Butte
r, x_Pork, x_Sardines_in_Oil, x_White_Tuna_in_Water, x_Popcorn,Air_Popped, x_Potato_
Chips,Bbqflvr, x_Pretzels, x_Tortilla_Chip, x_Chicknoodl_Soup, x_Splt_Pea&Hamsoup, x
_Vegetbeef_Soup, x_Neweng_Clamchwd, x_Tomato_Soup, x_New_E_Clamchwd,W_Mlk, x_Crm_Msh
rm_Soup,W_Mlk, x_Beanbacn_Soup,W_Watr]

Total Cost of Ingredients per day = $ 0.0
```

2.5 Solve another prob

In [19]:

```
prob = pulp.LpProblem('Another Diet prob', pulp.LpMinimize)


x = [0]*len(food_data)
for i, row in food_data.iterrows():
    x[i] = pulp.LpVariable(row['Foods'], 0, None, pulp.LpInteger)


prob += sum(food_data['Price/ Serving']*x)


for nutrient in food_columns[3:]:
    prob += sum(x*food_data[nutrient]) >= mins[nutrient]
    prob += sum(x*food_data[nutrient]) <= maxes[nutrient]


prob.writeLP("another_diet.lp")

prob.solve()
print(pulp.LpStatus[prob.status], '\n')


print('Diet consists of: \n')
for v in prob.variables():
    if v.varValue is not None and v.varValue > 0:
        print(v.name, '=', v.varValue)

print("\nTotal Cost of Ingredients per day = $",round(value(prob.objective),2))
```

```
Infeasible

Diet consists of:

Popcorn,Air_Popped = 0.069767442

Total Cost of Ingredients per day = $ 0.0
```

```
c:\Users\Clair\AppData\Local\Programs\Python\Python311\Lib\site-packages\pulp\pulp.p
y:1298: UserWarning: Spaces are not permitted in the name. Converted to '_'
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
```