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 project, 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.
2. Please add to your model the following constraints and solve the new model:
    a. If a food is selected, then a minimum of 1/10 serving must be chosen.
    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.

Methodology:

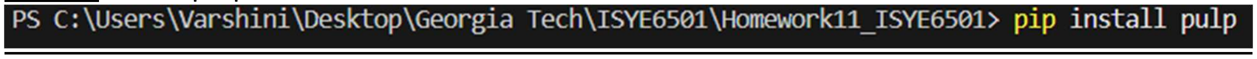
This linear programming model minimizes daily diet cost while satisfying nutritional requirements. The objective function minimizes total cost across all food servings. Constraints include:

1. meeting minimum and maximum bounds for 11 nutrients.
2. requiring at least 0.1 servings of any selected food (implemented via binary variables and the big-M method).
3. selecting exactly one of Celery or Frozen Broccoli
4. selecting exactly 3 meat/poultry/fish/egg items.

The model uses continuous variables for serving quantities and binary variables to track food selection.

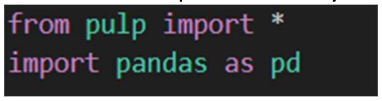
Solution:

Step-1: Install pulp in terminal.



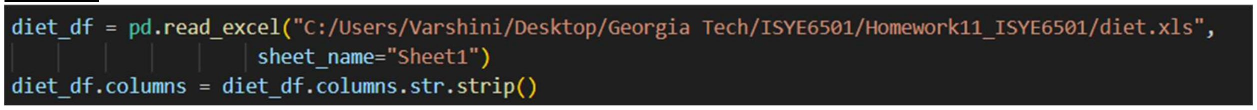
```
PS C:\Users\Varshini\Desktop\Georgia Tech\ISYE6501\Homework11_ISYE6501> pip install pulp
```

Then load required library in source code.



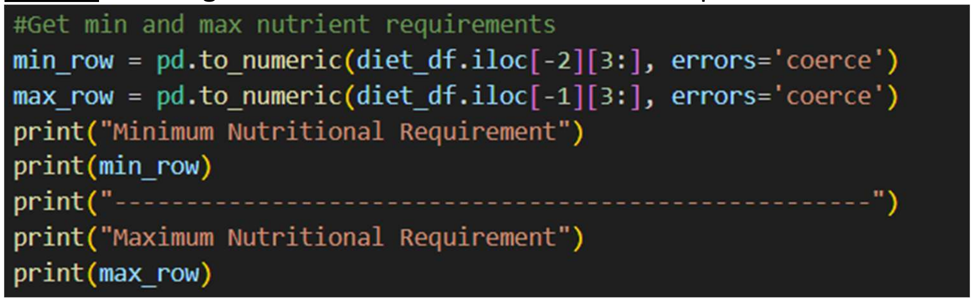
```
from pulp import *
import pandas as pd
```

Step-2: Load the data.



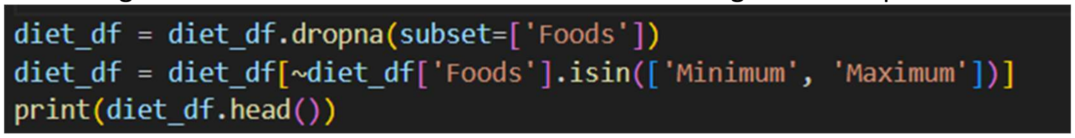
```
diet_df = pd.read_excel("C:/Users/Varshini/Desktop/Georgia Tech/ISYE6501/Homework11_ISYE6501/diet.xls",
                        sheet_name="Sheet1")
diet_df.columns = diet_df.columns.str.strip()
```

Step-3: Fetching minimum and maximum nutrient requirements from data.



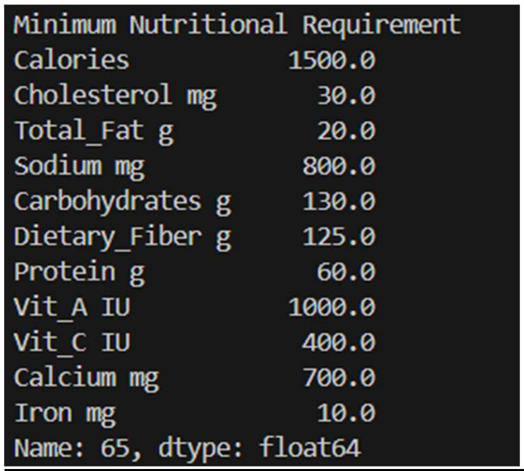
```
#Get min and max nutrient requirements
min_row = pd.to_numeric(diet_df.iloc[-2][3:], errors='coerce')
max_row = pd.to_numeric(diet_df.iloc[-1][3:], errors='coerce')
print("Minimum Nutritional Requirement")
print(min_row)
print("-------------------------------------------------------")
print("Maximum Nutritional Requirement")
print(max_row)
```

Removing min and max data from dataset. then checking out the top data.

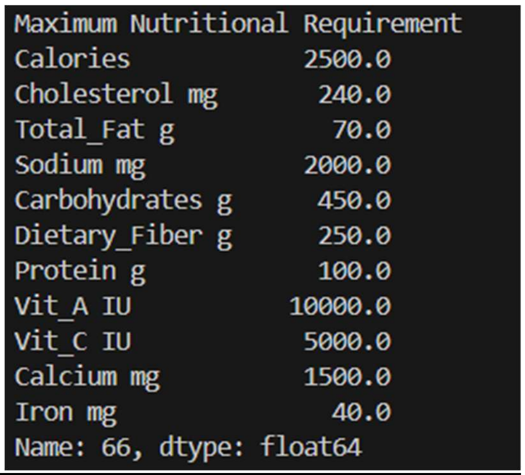


```
diet_df = diet_df.dropna(subset=['Foods'])
diet_df = diet_df[~diet_df['Foods'].isin(['Minimum', 'Maximum'])]
print(diet_df.head())
```

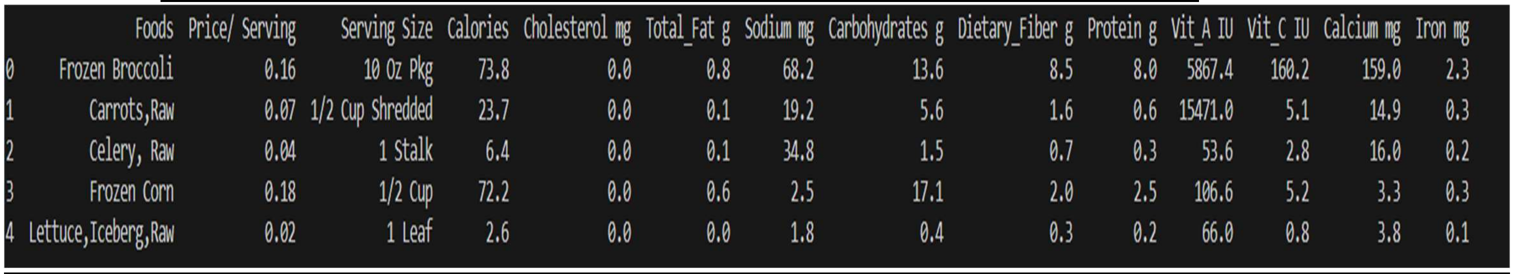
Output:



```
Minimum Nutritional Requirement
Calories              1500.0
Cholesterol mg          30.0
Total_Fat g             20.0
Sodium mg              800.0
Carbohydrates g        130.0
Dietary_Fiber g        125.0
Protein g               60.0
Vit_A IU              1000.0
Vit_C IU               400.0
Calcium mg             700.0
Iron mg                 10.0
Name: 65, dtype: float64
```

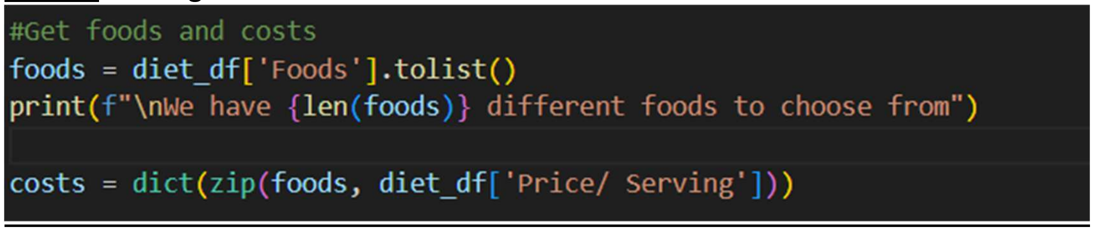


```
Maximum Nutritional Requirement
Calories              2500.0
Cholesterol mg         240.0
Total_Fat g             70.0
Sodium mg             2000.0
Carbohydrates g        450.0
Dietary_Fiber g        250.0
Protein g              100.0
Vit_A IU             10000.0
Vit_C IU              5000.0
Calcium mg            1500.0
Iron mg                 40.0
Name: 66, dtype: float64
```



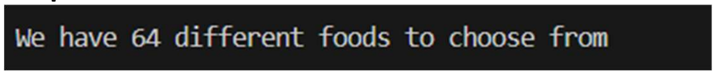
```
               Foods  Price/ Serving        Serving Size  Calories  Cholesterol mg  Total_Fat g  Sodium mg  Carbohydrates g  Dietary_Fiber g  Protein g  Vit_A IU  Vit_C IU  Calcium mg  Iron mg
0    Frozen Broccoli            0.16          10 Oz Pkg      73.8             0.0          0.8       68.2             13.6              8.5        8.0    5867.4     160.2       159.0      2.3
1         Carrots,Raw            0.07  1/2 Cup Shredded      23.7             0.0          0.1       19.2              5.6              1.6        0.6   15471.0       5.1        14.9      0.3
2         Celery, Raw            0.04            1 Stalk       6.4             0.0          0.1       34.8              1.5              0.7        0.3      53.6       2.8        16.0      0.2
3         Frozen Corn            0.18            1/2 Cup      72.2             0.0          0.6        2.5             17.1              2.0        2.5     106.6       5.2         3.3      0.3
4  Lettuce,Iceberg,Raw           0.02             1 Leaf       2.6             0.0          0.0        1.8              0.4              0.3        0.2      66.0       0.8         3.8      0.1
```

Step-4: Getting list of foods and their costs.



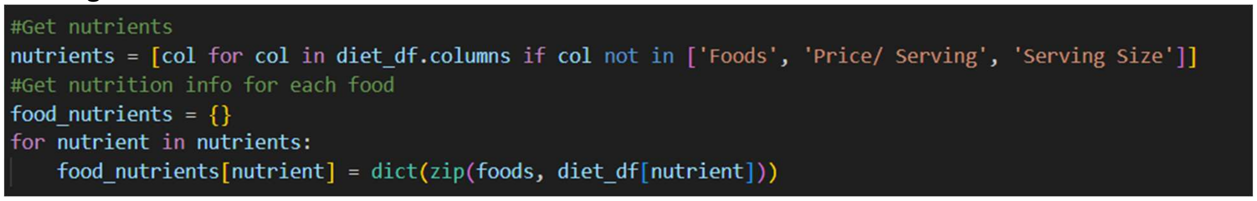
```
#Get foods and costs
foods = diet_df['Foods'].tolist()
print(f"\nWe have {len(foods)} different foods to choose from")

costs = dict(zip(foods, diet_df['Price/ Serving']))
```

Output:



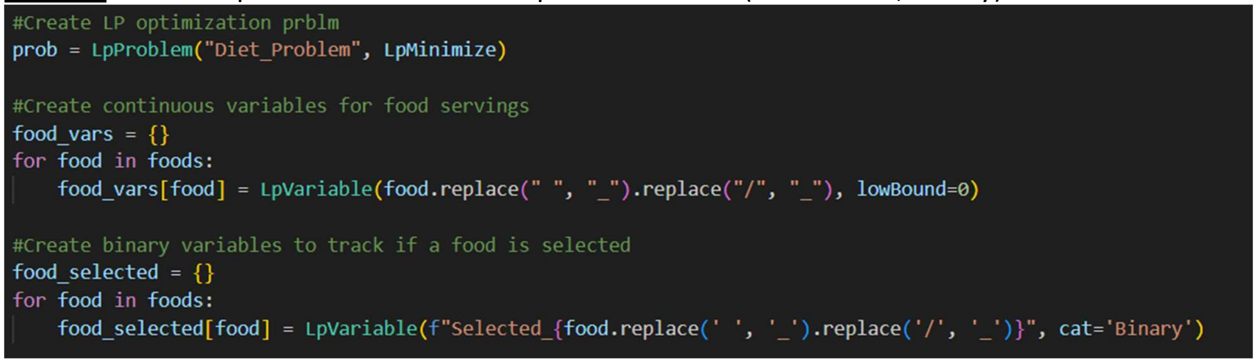
```
We have 64 different foods to choose from
```

Getting list of nutrients from columns.



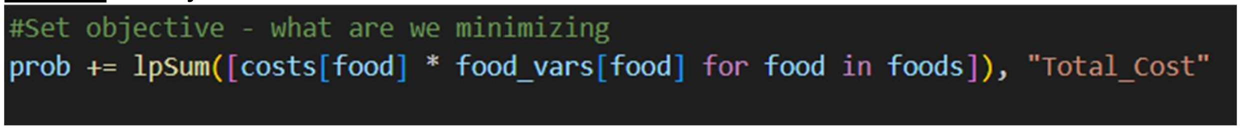
```
#Get nutrients
nutrients = [col for col in diet_df.columns if col not in ['Foods', 'Price/ Serving', 'Serving Size']]
#Get nutrition info for each food
food_nutrients = {}
for nutrient in nutrients:
    food_nutrients[nutrient] = dict(zip(foods, diet_df[nutrient]))
```

Step-5: Create LP problem and create required variables (continuous, binary).



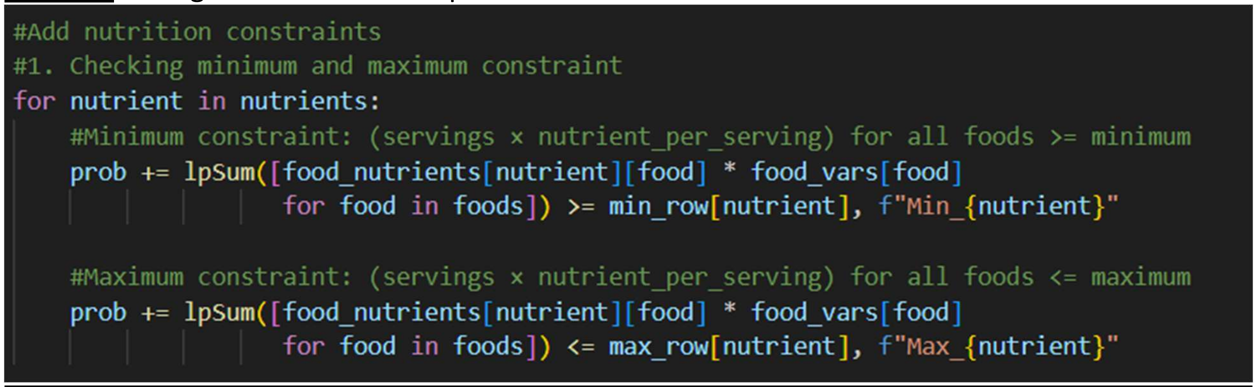
```
#Create LP optimization prblm
prob = LpProblem("Diet_Problem", LpMinimize)

#Create continuous variables for food servings
food_vars = {}
for food in foods:
    food_vars[food] = LpVariable(food.replace(" ", "_").replace("/", "_"), lowBound=0)

#Create binary variables to track if a food is selected
food_selected = {}
for food in foods:
    food_selected[food] = LpVariable(f"Selected_{food.replace(' ', '_').replace('/', '_')}", cat='Binary')
```

Step-6: Set objective

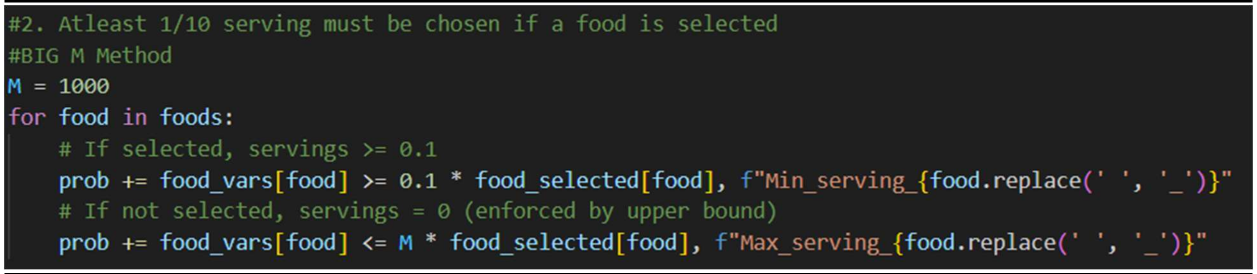


```
#Set objective - what are we minimizing
prob += lpSum([costs[food] * food_vars[food] for food in foods]), "Total_Cost"
```

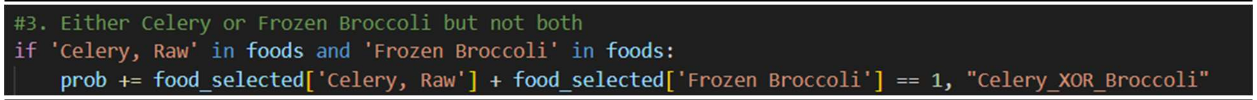
Step-7: Adding constraints to the problem.



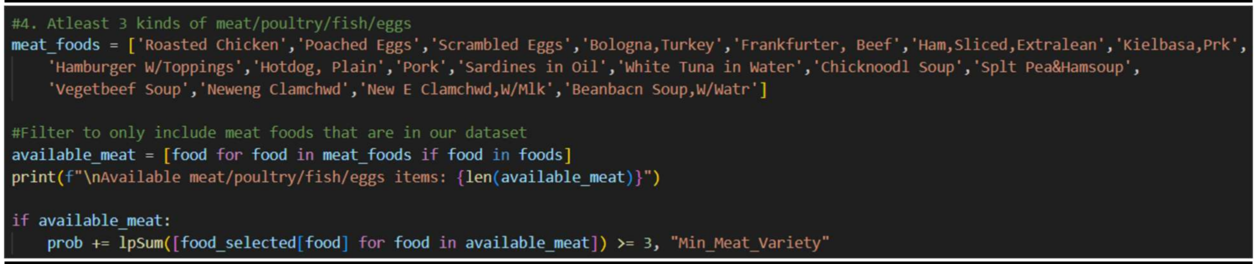
```
#Add nutrition constraints
#1. Checking minimum and maximum constraint
for nutrient in nutrients:
    #Minimum constraint: (servings × nutrient_per_serving) for all foods >= minimum
    prob += lpSum([food_nutrients[nutrient][food] * food_vars[food]
                   for food in foods]) >= min_row[nutrient], f"Min_{nutrient}"

    #Maximum constraint: (servings × nutrient_per_serving) for all foods <= maximum
    prob += lpSum([food_nutrients[nutrient][food] * food_vars[food]
                   for food in foods]) <= max_row[nutrient], f"Max_{nutrient}"
```



```
#2. Atleast 1/10 serving must be chosen if a food is selected
#BIG M Method
M = 1000
for food in foods:
    # If selected, servings >= 0.1
    prob += food_vars[food] >= 0.1 * food_selected[food], f"Min_serving_{food.replace(' ', '_')}"
    # If not selected, servings = 0 (enforced by upper bound)
    prob += food_vars[food] <= M * food_selected[food], f"Max_serving_{food.replace(' ', '_')}"
```

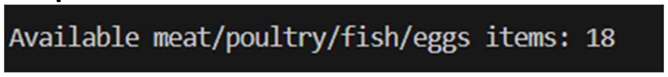


```
#3. Either Celery or Frozen Broccoli but not both
if 'Celery, Raw' in foods and 'Frozen Broccoli' in foods:
    prob += food_selected['Celery, Raw'] + food_selected['Frozen Broccoli'] == 1, "Celery_XOR_Broccoli"
```



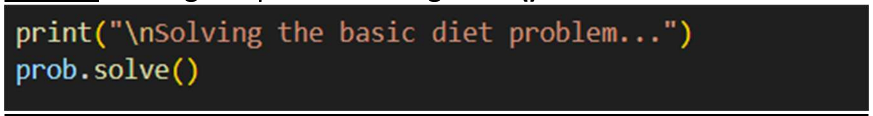
```
#4. Atleast 3 kinds of meat/poultry/fish/eggs
meat_foods = ['Roasted Chicken','Poached Eggs','Scrambled Eggs','Bologna,Turkey','Frankfurter, Beef','Ham,Sliced,Extralean','Kielbasa,Prk',
    'Hamburger W/Toppings','Hotdog, Plain','Pork','Sardines in Oil','White Tuna in Water','Chicknoodl Soup','Splt Pea&Hamsoup',
    'Vegetbeef Soup','Neweng Clamchwd','New E Clamchwd,W/Mlk','Beanbacn Soup,W/Watr']

#Filter to only include meat foods that are in our dataset
available_meat = [food for food in meat_foods if food in foods]
print(f"\nAvailable meat/poultry/fish/eggs items: {len(available_meat)}")

if available_meat:
    prob += lpSum([food_selected[food] for food in available_meat]) >= 3, "Min_Meat_Variety"
```

Output:

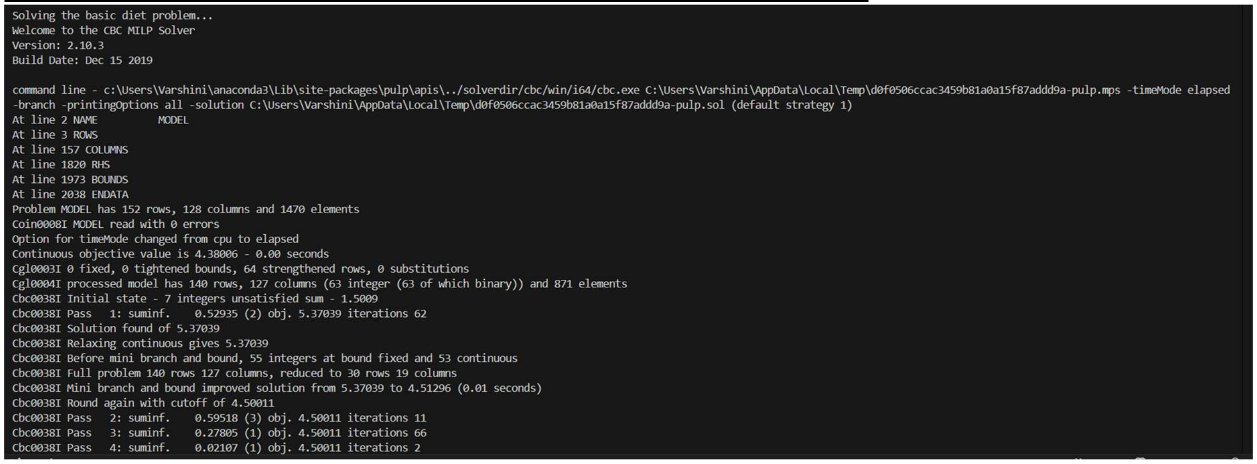


```
Available meat/poultry/fish/eggs items: 18
```

Step-8: Solving the problem using **solve()**.

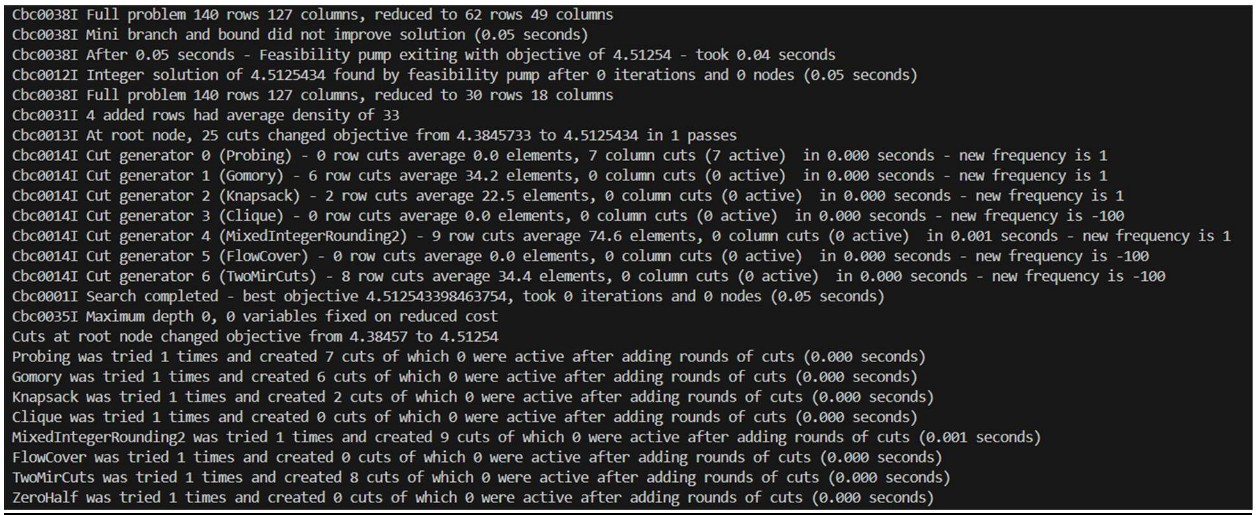


```python
print("\nSolving the basic diet problem...")
prob.solve()
```



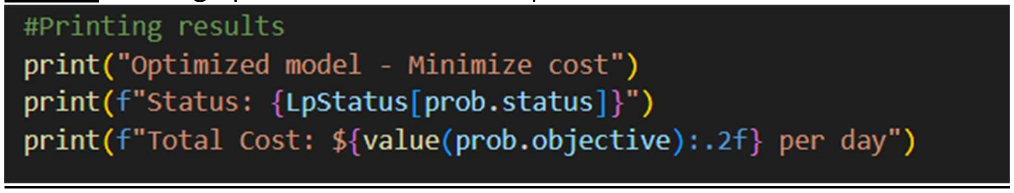
```
Solving the basic diet problem...
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - c:\Users\Varshini\anaconda3\Lib\site-packages\pulp\apis\../solverdir/cbc/win/i64/cbc.exe C:\Users\Varshini\AppData\Local\Temp\d0f0506ccac3459b81a0a15f87addd9a-pulp.mps -timeMode elapsed
-branch -printingOptions all -solution C:\Users\Varshini\AppData\Local\Temp\d0f0506ccac3459b81a0a15f87addd9a-pulp.sol (default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 157 COLUMNS
At line 1820 RHS
At line 1973 BOUNDS
At line 2038 ENDATA
Problem MODEL has 152 rows, 128 columns and 1470 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Continuous objective value is 4.38006 - 0.00 seconds
Cgl0003I 0 fixed, 0 tightened bounds, 64 strengthened rows, 0 substitutions
Cgl0004I processed model has 140 rows, 127 columns (63 integer (63 of which binary)) and 871 elements
Cbc0038I Initial state - 7 integers unsatisfied sum - 1.5009
Cbc0038I Pass   1: suminf.    0.52935 (2) obj. 5.37039 iterations 62
Cbc0038I Solution found of 5.37039
Cbc0038I Relaxing continuous gives 5.37039
Cbc0038I Before mini branch and bound, 55 integers at bound fixed and 53 continuous
Cbc0038I Full problem 140 rows 127 columns, reduced to 30 rows 19 columns
Cbc0038I Mini branch and bound improved solution from 5.37039 to 4.51296 (0.01 seconds)
Cbc0038I Round again with cutoff of 4.50011
Cbc0038I Pass   2: suminf.    0.59518 (3) obj. 4.50011 iterations 11
Cbc0038I Pass   3: suminf.    0.27805 (1) obj. 4.50011 iterations 66
Cbc0038I Pass   4: suminf.    0.02107 (1) obj. 4.50011 iterations 2
```

…

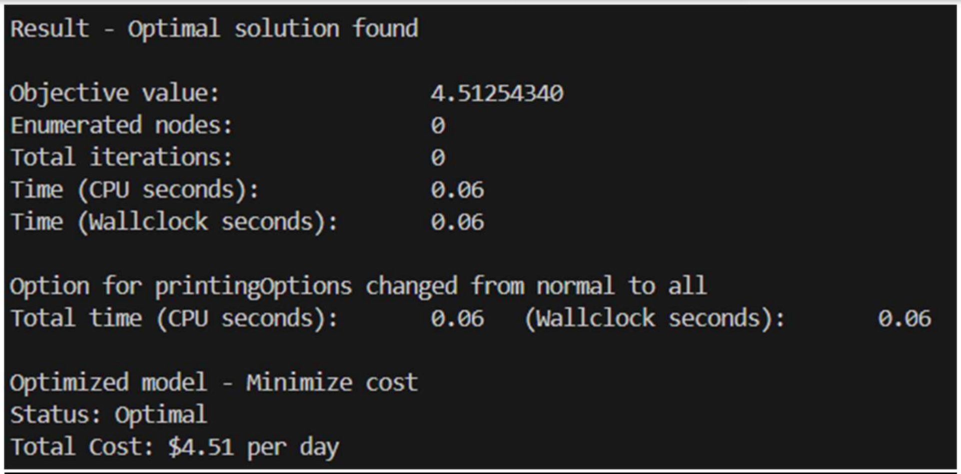


```
Cbc0038I Full problem 140 rows 127 columns, reduced to 62 rows 49 columns
Cbc0038I Mini branch and bound did not improve solution (0.05 seconds)
Cbc0038I After 0.05 seconds - Feasibility pump exiting with objective of 4.51254 - took 0.04 seconds
Cbc0012I Integer solution of 4.5125434 found by feasibility pump after 0 iterations and 0 nodes (0.05 seconds)
Cbc0038I Full problem 140 rows 127 columns, reduced to 30 rows 18 columns
Cbc0031I 4 added rows had average density of 33
Cbc0013I At root node, 25 cuts changed objective from 4.3845733 to 4.5125434 in 1 passes
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 7 column cuts (7 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 6 row cuts average 34.2 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 2 (Knapsack) - 2 row cuts average 22.5 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) - 9 row cuts average 74.6 elements, 0 column cuts (0 active)  in 0.001 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) - 8 row cuts average 34.4 elements, 0 column cuts (0 active)  in 0.000 seconds - new frequency is -100
Cbc0001I Search completed - best objective 4.512543398463754, took 0 iterations and 0 nodes (0.05 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from 4.38457 to 4.51254
Probing was tried 1 times and created 7 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 1 times and created 6 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 1 times and created 2 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 1 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 1 times and created 9 cuts of which 0 were active after adding rounds of cuts (0.001 seconds)
FlowCover was tried 1 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 1 times and created 8 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 1 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
```

Step-9: Printing optimal solution for the problem based on the variables and constraints.

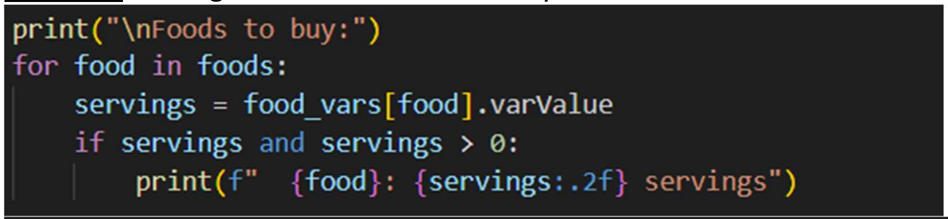


```python
#Printing results
print("Optimized model - Minimize cost")
print(f"Status: {LpStatus[prob.status]}")
print(f"Total Cost: ${value(prob.objective):.2f} per day")
```

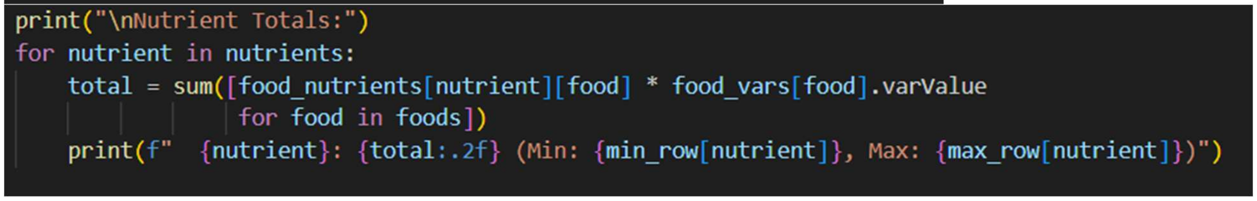


```
Result - Optimal solution found

Objective value:                4.51254340
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.06
Time (Wallclock seconds):       0.06

Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.06   (Wallclock seconds):       0.06

Optimized model - Minimize cost
Status: Optimal
Total Cost: $4.51 per day
```

Step-10: Printing the results – what to buy and how much nutrients it has.

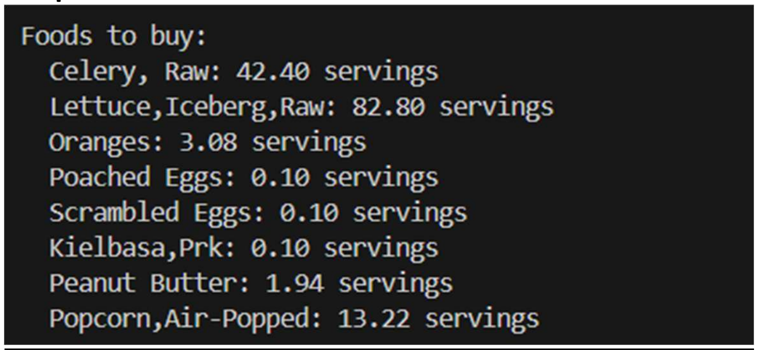


```python
print("\nFoods to buy:")
for food in foods:
    servings = food_vars[food].varValue
    if servings and servings > 0:
        print(f"  {food}: {servings:.2f} servings")
```

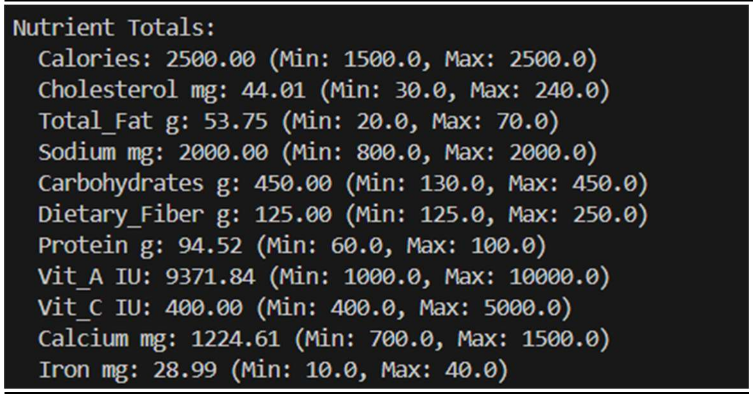


```python
print("\nNutrient Totals:")
for nutrient in nutrients:
    total = sum([food_nutrients[nutrient][food] * food_vars[food].varValue
                 for food in foods])
    print(f"  {nutrient}: {total:.2f} (Min: {min_row[nutrient]}, Max: {max_row[nutrient]})")
```

Output:



```
Foods to buy:
  Celery, Raw: 42.40 servings
  Lettuce,Iceberg,Raw: 82.80 servings
  Oranges: 3.08 servings
  Poached Eggs: 0.10 servings
  Scrambled Eggs: 0.10 servings
  Kielbasa,Prk: 0.10 servings
  Peanut Butter: 1.94 servings
  Popcorn,Air-Popped: 13.22 servings
```



```
Nutrient Totals:
  Calories: 2500.00 (Min: 1500.0, Max: 2500.0)
  Cholesterol mg: 44.01 (Min: 30.0, Max: 240.0)
  Total_Fat g: 53.75 (Min: 20.0, Max: 70.0)
  Sodium mg: 2000.00 (Min: 800.0, Max: 2000.0)
  Carbohydrates g: 450.00 (Min: 130.0, Max: 450.0)
  Dietary_Fiber g: 125.00 (Min: 125.0, Max: 250.0)
  Protein g: 94.52 (Min: 60.0, Max: 100.0)
  Vit_A IU: 9371.84 (Min: 1000.0, Max: 10000.0)
  Vit_C IU: 400.00 (Min: 400.0, Max: 5000.0)
  Calcium mg: 1224.61 (Min: 700.0, Max: 1500.0)
  Iron mg: 28.99 (Min: 10.0, Max: 40.0)
```

Result Discussion:
The optimal solution costs $4.51 per day and selects 8 foods.

1. The diet is dominated by inexpensive vegetables—42.4 servings of celery and 82.8 servings of lettuce—which efficiently meet multiple nutritional constraints at low cost.
2. The model satisfies the celery/broccoli constraint by selecting only celery.
3. It meets the protein requirement by choosing exactly 3 meat types at minimum servings (0.1 each): poached eggs, scrambled eggs, and kielbasa.
4. All 11 nutrients fall within required bounds, with several (calories, sodium, carbohydrates, dietary fiber, vitamin C) at their maximum limits, indicating these are the binding constraints.

While mathematically optimal, the solution is impractical due to unrealistic serving quantities, suggesting additional constraints (e.g., maximum servings per food, variety requirements) would be needed for real-world application.