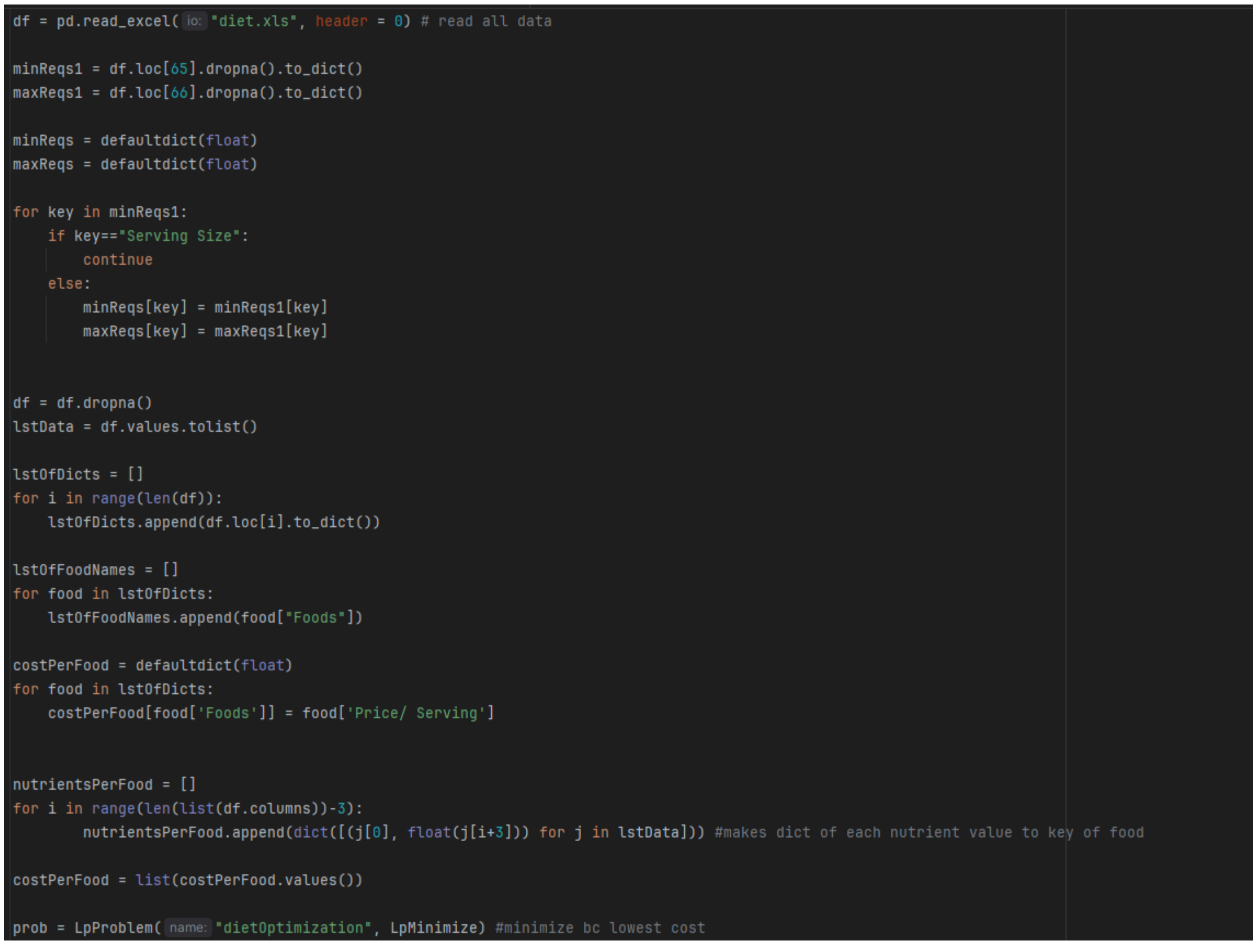
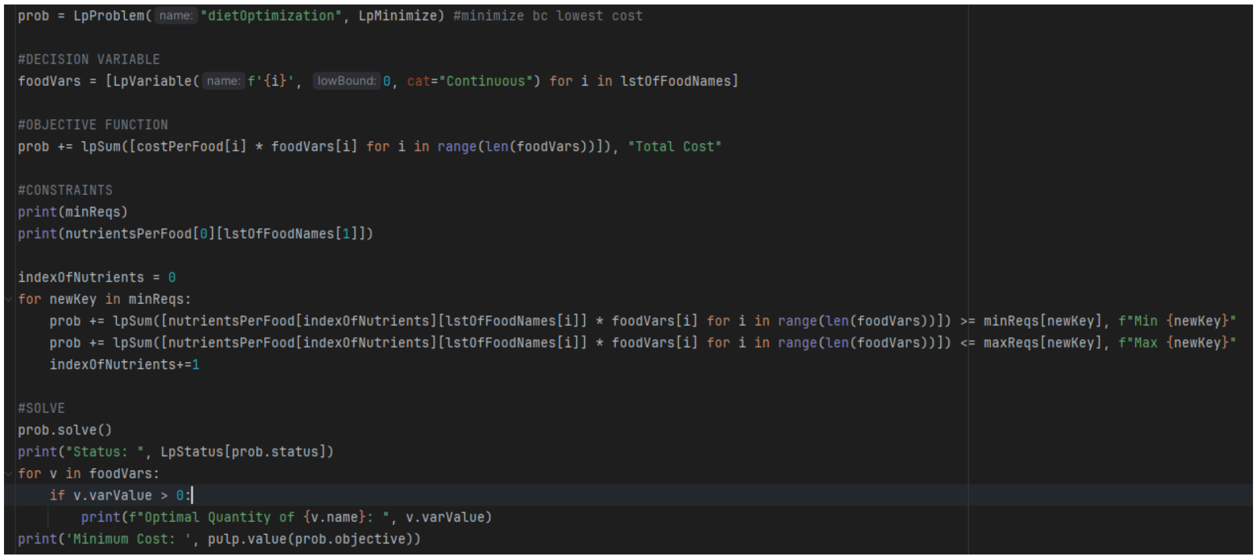


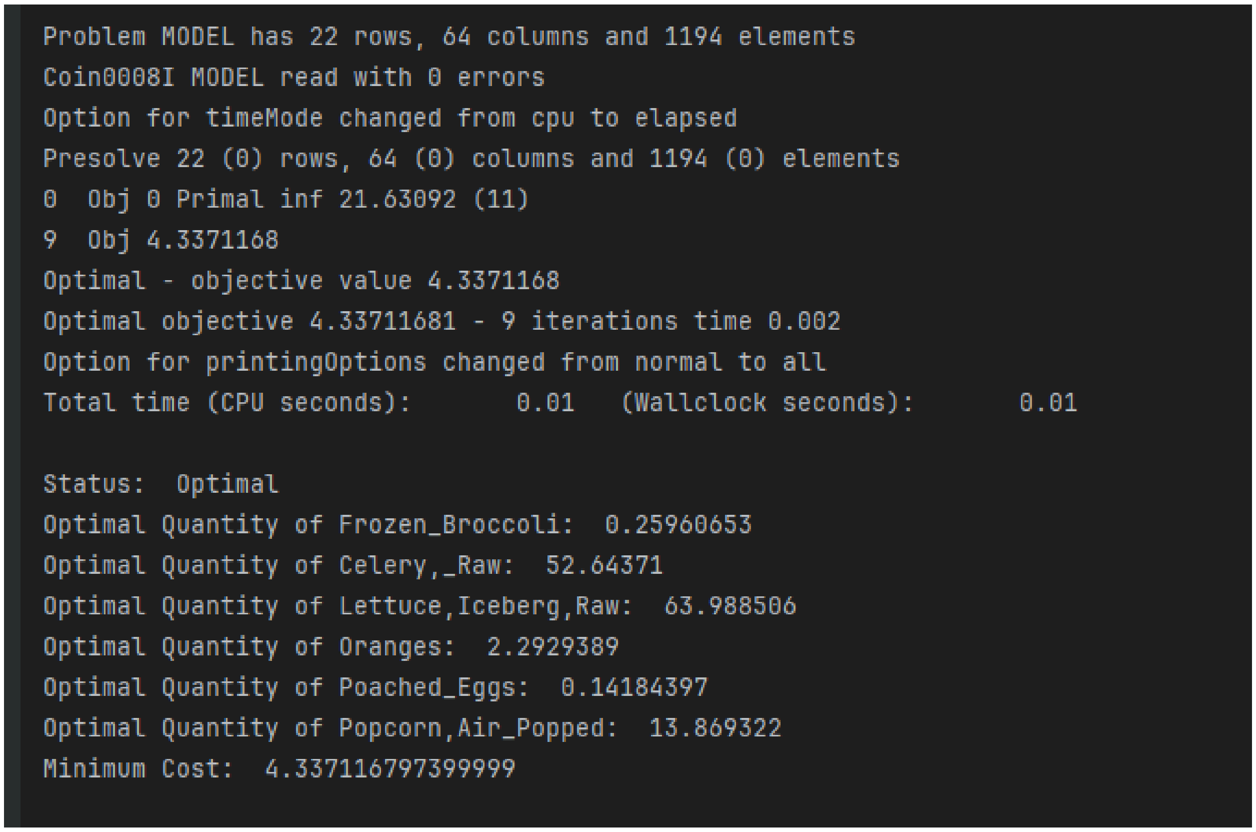
```python
df = pd.read_excel( io: "diet.xls", header = 0) # read all data

minReqs1 = df.loc[65].dropna().to_dict()
maxReqs1 = df.loc[66].dropna().to_dict()

minReqs = defaultdict(float)
maxReqs = defaultdict(float)

for key in minReqs1:
    if key=="Serving Size":
        continue
    else:
        minReqs[key] = minReqs1[key]
        maxReqs[key] = maxReqs1[key]


df = df.dropna()
lstData = df.values.tolist()

lstOfDicts = []
for i in range(len(df)):
    lstOfDicts.append(df.loc[i].to_dict())

lstOfFoodNames = []
for food in lstOfDicts:
    lstOfFoodNames.append(food["Foods"])

costPerFood = defaultdict(float)
for food in lstOfDicts:
    costPerFood[food['Foods']] = food['Price/ Serving']


nutrientsPerFood = []
for i in range(len(list(df.columns))-3):
        nutrientsPerFood.append(dict([(j[0], float(j[i+3])) for j in lstData])) #makes dict of each nutrient value to key of food

costPerFood = list(costPerFood.values())

prob = LpProblem( name: "dietOptimization", LpMinimize) #minimize bc lowest cost
```

**Figure 1.** This is the code I used to unpack and sort the data from the xls file given. I utilized pandas and xlrd to process the data. I dropped all rows containing NaN values so that when I iterated through I would be able to ascertain all of the nutritional factors of the foods present. Each variable is aptly named to what it describes and there are comments for the more unintuitive portions of the code.



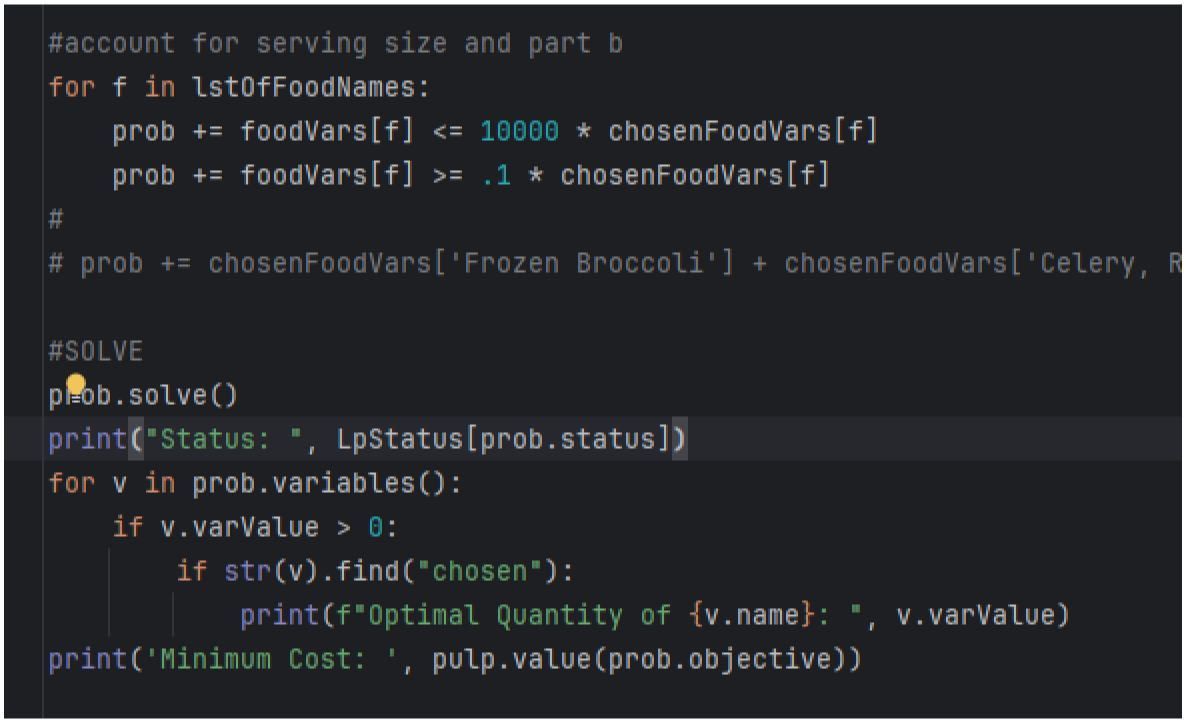
```python
prob = LpProblem( name: "dietOptimization", LpMinimize) #minimize bc lowest cost

#DECISION VARIABLE
foodVars = [LpVariable( name: f'{i}',  lowBound: 0, cat="Continuous") for i in lstOfFoodNames]

#OBJECTIVE FUNCTION
prob += lpSum([costPerFood[i] * foodVars[i] for i in range(len(foodVars))]), "Total Cost"

#CONSTRAINTS
print(minReqs)
print(nutrientsPerFood[0][lstOfFoodNames[1]])

indexOfNutrients = 0
for newKey in minReqs:
    prob += lpSum([nutrientsPerFood[indexOfNutrients][lstOfFoodNames[i]] * foodVars[i] for i in range(len(foodVars))]) >= minReqs[newKey], f"Min {newKey}"
    prob += lpSum([nutrientsPerFood[indexOfNutrients][lstOfFoodNames[i]] * foodVars[i] for i in range(len(foodVars))]) <= maxReqs[newKey], f"Max {newKey}"
    indexOfNutrients+=1

#SOLVE
prob.solve()
print("Status: ", LpStatus[prob.status])
for v in foodVars:
    if v.varValue > 0:
        print(f"Optimal Quantity of {v.name}: ", v.varValue)
print('Minimum Cost: ', pulp.value(prob.objective))
```

**Figure 2.** This figure denotes the three components of the optimization problem in question 1. Most of this code was structured via the office hour lesson on PuLP. There are comments in the code detailing each step of the process towards completing this assignment.



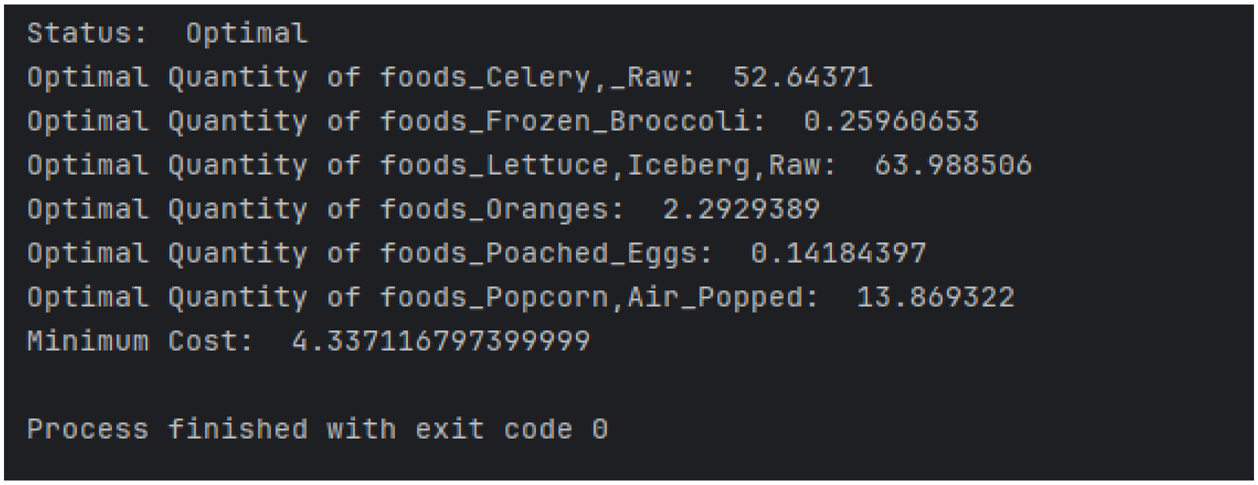
```
Problem MODEL has 22 rows, 64 columns and 1194 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 22 (0) rows, 64 (0) columns and 1194 (0) elements
0  Obj 0 Primal inf 21.63092 (11)
9  Obj 4.3371168
Optimal - objective value 4.3371168
Optimal objective 4.33711681 - 9 iterations time 0.002
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.01   (Wallclock seconds):       0.01

Status:  Optimal
Optimal Quantity of Frozen_Broccoli:  0.25960653
Optimal Quantity of Celery,_Raw:  52.64371
Optimal Quantity of Lettuce,Iceberg,Raw:  63.988506
Optimal Quantity of Oranges:  2.2929389
Optimal Quantity of Poached_Eggs:  0.14184397
Optimal Quantity of Popcorn,Air_Popped:  13.869322
Minimum Cost:  4.337116797399999
```

**Figure 3.** This figure is the output of the above code in Figure 2. The final cost of the food plan comes out at around $4.34. Notice that this combination lines up exactly with what was expected in question 1.



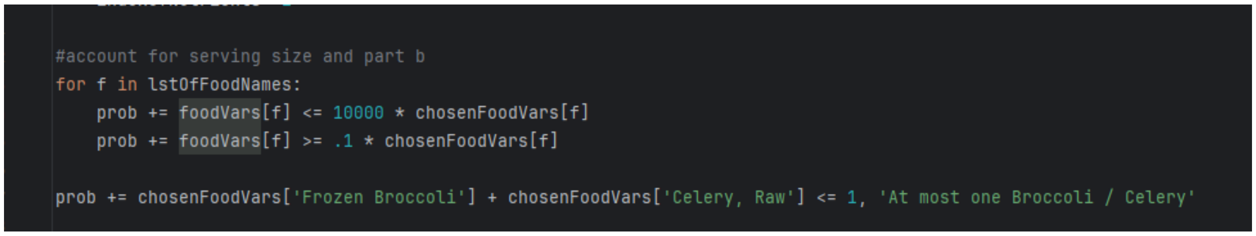
```
#account for serving size and part b
for f in lstOfFoodNames:
    prob += foodVars[f] <= 10000 * chosenFoodVars[f]
    prob += foodVars[f] >= .1 * chosenFoodVars[f]
#
# prob += chosenFoodVars['Frozen Broccoli'] + chosenFoodVars['Celery, R

#SOLVE
prob.solve()
print("Status: ", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue > 0:
        if str(v).find("chosen"):
            print(f"Optimal Quantity of {v.name}: ", v.varValue)
print('Minimum Cost: ', pulp.value(prob.objective))
```

**Figure 5.** This along with the line: `foodVars = LpVariable.dicts('foods', lstOfFoodNames, 0, cat="Continuous") chosenFoodVars = LpVariable.dicts("chosen",lstOfFoodNames,lowBound=0, upBound=1, cat="Binary")`, was added into the same model ran above in Figures 1 and 2, in order to answer question 2a. I realized after countless google searches that applying the dicts function to the variables was the only way to get this solution to work. I am not sure why, but I kept getting the name (i.e string) whenever I tried to access the binary variable until I made it into a dictionary. It is also important to note that inorder to get this code to execute, I edited the values in the serving size of the xls data to be just numbers, rather than have them be accompanied by units.



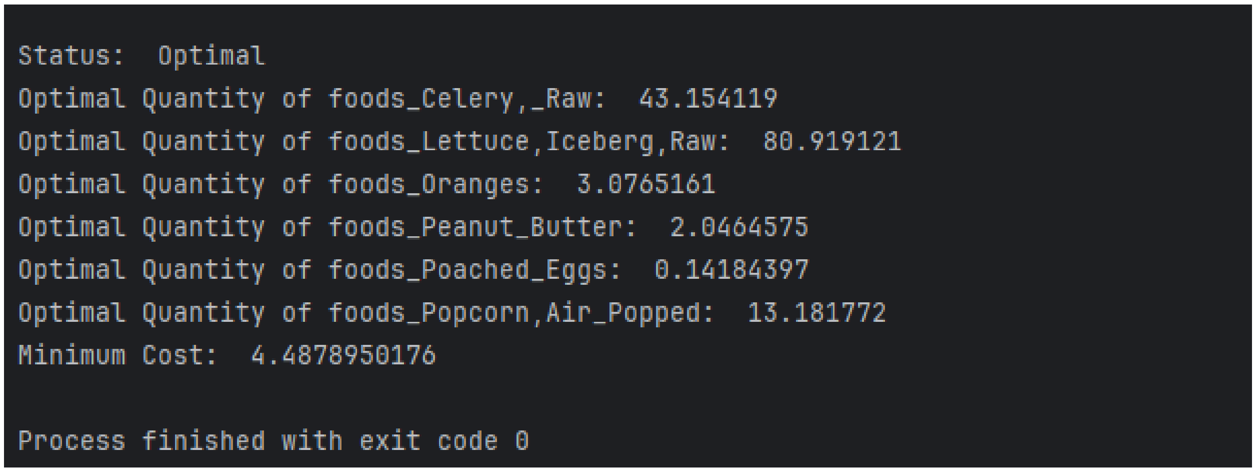
```
Status:  Optimal
Optimal Quantity of foods_Celery,_Raw:  52.64371
Optimal Quantity of foods_Frozen_Broccoli:  0.25960653
Optimal Quantity of foods_Lettuce,Iceberg,Raw:  63.988506
Optimal Quantity of foods_Oranges:  2.2929389
Optimal Quantity of foods_Poached_Eggs:  0.14184397
Optimal Quantity of foods_Popcorn,Air_Popped:  13.869322
Minimum Cost:  4.337116797399999

Process finished with exit code 0
```

**Figure 6.** This is the output generated with the above code. It is not different in any way and that makes sense because of the serving size being greater than 0.10 for all of them in the prior optimal solution.



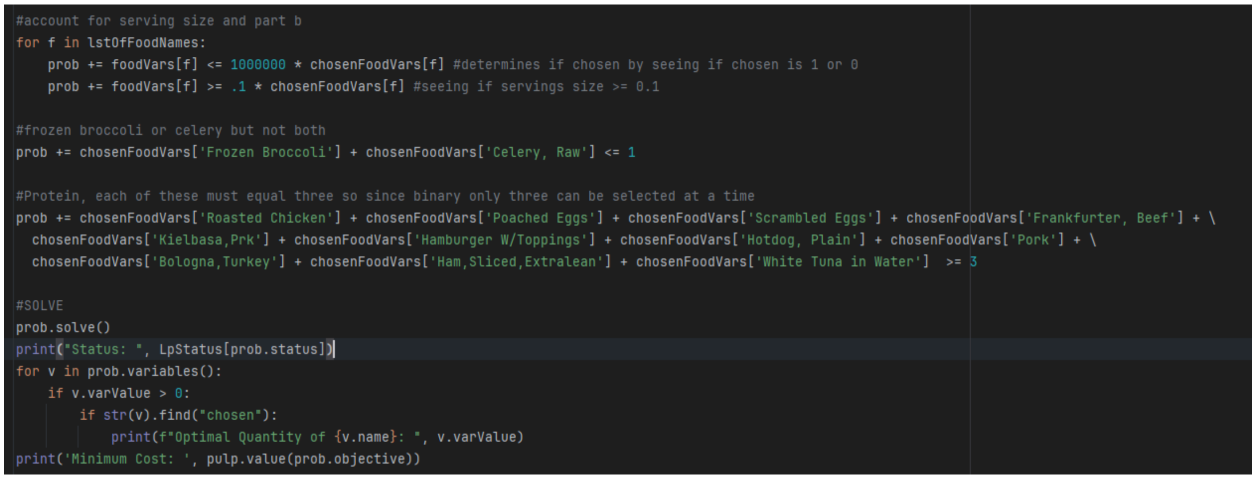
```
#account for serving size and part b
for f in lstOfFoodNames:
    prob += foodVars[f] <= 10000 * chosenFoodVars[f]
    prob += foodVars[f] >= .1 * chosenFoodVars[f]

prob += chosenFoodVars['Frozen Broccoli'] + chosenFoodVars['Celery, Raw'] <= 1, 'At most one Broccoli / Celery'
```

**Figure 7.** This is the code used to generate the values after applying the constraint listed in 2b. Again, I kept getting strings returned to me when I did not make the variables dicts, so if you have any insight as to why that would happen, please let me know in the comments.



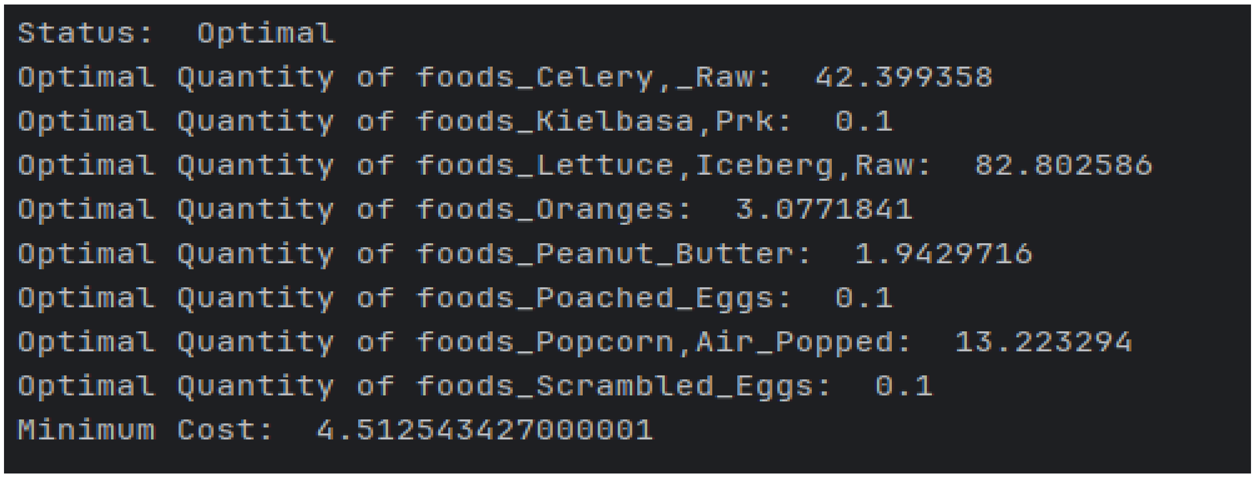
```
Status:  Optimal
Optimal Quantity of foods_Celery,_Raw:  43.154119
Optimal Quantity of foods_Lettuce,Iceberg,Raw:  80.919121
Optimal Quantity of foods_Oranges:  3.0765161
Optimal Quantity of foods_Peanut_Butter:  2.0464575
Optimal Quantity of foods_Poached_Eggs:  0.14184397
Optimal Quantity of foods_Popcorn,Air_Popped:  13.181772
Minimum Cost:  4.4878950176

Process finished with exit code 0
```

**Figure 8.** This is the output of the code in Figure 7 to answer part 2b. This is the first different answer compared to the prior outputs. Because of the fact that celery is present and frozen broccoli is not, the code shows that it is running as it should.



```
#account for serving size and part b
for f in lstOfFoodNames:
    prob += foodVars[f] <= 1000000 * chosenFoodVars[f] #determines if chosen by seeing if chosen is 1 or 0
    prob += foodVars[f] >= .1 * chosenFoodVars[f] #seeing if servings size >= 0.1

#frozen broccoli or celery but not both
prob += chosenFoodVars['Frozen Broccoli'] + chosenFoodVars['Celery, Raw'] <= 1

#Protein, each of these must equal three so since binary only three can be selected at a time
prob += chosenFoodVars['Roasted Chicken'] + chosenFoodVars['Poached Eggs'] + chosenFoodVars['Scrambled Eggs'] + chosenFoodVars['Frankfurter, Beef'] + \
  chosenFoodVars['Kielbasa,Prk'] + chosenFoodVars['Hamburger W/Toppings'] + chosenFoodVars['Hotdog, Plain'] + chosenFoodVars['Pork'] + \
  chosenFoodVars['Bologna,Turkey'] + chosenFoodVars['Ham,Sliced,Extralean'] + chosenFoodVars['White Tuna in Water']  >= 3

#SOLVE
prob.solve()
print("Status: ", LpStatus[prob.status])
for v in prob.variables():
    if v.varValue > 0:
        if str(v).find("chosen"):
            print(f"Optimal Quantity of {v.name}: ", v.varValue)
print('Minimum Cost: ', pulp.value(prob.objective))
```

**Figure 9.** This is the code used to generate the output for question 2c, which will subsequently follow in Figure 10. The comment above the new added line explains the thought process behind why that was used.



```
Status:  Optimal
Optimal Quantity of foods_Celery,_Raw:  42.399358
Optimal Quantity of foods_Kielbasa,Prk:  0.1
Optimal Quantity of foods_Lettuce,Iceberg,Raw:  82.802586
Optimal Quantity of foods_Oranges:  3.0771841
Optimal Quantity of foods_Peanut_Butter:  1.9429716
Optimal Quantity of foods_Poached_Eggs:  0.1
Optimal Quantity of foods_Popcorn,Air_Popped:  13.223294
Optimal Quantity of foods_Scrambled_Eggs:  0.1
Minimum Cost:  4.512543427000001
```

**Figure 10.** This final figure is the output associated with the cumulative constraints from each part of question 2. The minimum cost associated with this optimization is about $4.51.