In [95]:
```
import pandas as pd
import numpy as np
```

Preprocessing

In [96]:
```
df = pd.read_csv("nutrition 2.csv")
df.drop(columns=['Unnamed: 0','lucopene', 'water'], inplace=True)
```

In [97]:
```
df.head()
df.rename(columns={'zink': 'zinc'}, inplace=True)
df.rename(columns={'irom': 'iron'}, inplace=True)
```

In [98]:
```
def remove_units_and_convert(value):
    if pd.isna(value):
        return 0
    if isinstance(value, str):
        try:
            return float(''.join(filter(lambda x: x.isdigit() or x == '.', v
        except ValueError:
            return 0
    else:
        return value

# Applying the function to all the columns except 'name', 'serving_size', ar
for col in df.columns:
    if col not in ['name', 'Unnamed: 0']:
        df[col] = df[col].apply(remove_units_and_convert)

# Renaming the columns to include unit
new_column_names = {}
for col in df.columns:
    if col in [ 'vitamin_e',
       'tocopherol_alpha', 'fatty_acids_total_trans','caffeine' ,'theobromi
        new_column_names[col] = col + ' (mg)'
    elif col in ['fructose', 'galactose', 'glucose', 'lactose', 'maltose',
       'arginine', 'aspartic_acid', 'cystine', 'glutamic_acid', 'glycine',
       'histidine', 'hydroxyproline', 'isoleucine', 'leucine', 'lysine',
       'methionine', 'phenylalanine', 'proline', 'serine', 'threonine',
       'tryptophan', 'tyrosine', 'valine']:
        new_column_names[col] = col + ' (g)'
    elif col in ['calories']:
        new_column_names[col] = col + ' (kcal)'
    elif col in [ 'vitamin_a_rae', 'folate',
       'carotene_alpha', 'carotene_beta', 'cryptoxanthin_beta',
       'lutein_zeaxanthin','folic_acid', "selenium", 'vitamin_k']:
        new_column_names[col] = col + ' (mcg)'
    elif col in ['vitamin_a', 'vitamin_d']:
        new_column_names[col] = col + ' (IU)'
    else:
        new_column_names[col] = col  # For columns that don't need a unit

df.rename(columns=new_column_names, inplace=True)
```

In [99]:
```
df
```

Out[99]:

	name	serving_size (g)	calories (kcal)	total_fat (g)	saturated_fat (g)	cholesterol (mg)	sodium (mg)	choli (m
0	Cornstarch	100.0	381	0.1	0.0	0.0	9.0	(
1	Nuts, pecans	100.0	691	72.0	6.2	0.0	0.0	4(
2	Eggplant, raw	100.0	25	0.2	0.0	0.0	2.0	6
3	Teff, uncooked	100.0	367	2.4	0.4	0.0	12.0	1
4	Sherbet, orange	100.0	144	2.0	1.2	1.0	46.0	
...	...	...	...	...	...	...	...	
8784	Beef, raw, all grades, trimmed to 0" fat, sepa...	100.0	125	3.5	1.4	62.0	54.0	6
8785	Lamb, cooked, separable lean only, composite o...	100.0	206	8.9	3.9	109.0	50.0	(
8786	Lamb, raw, separable lean and fat, composite o...	100.0	277	23.0	12.0	78.0	39.0	(
8787	Beef, raw, all grades, trimmed to 0" fat, sepa...	100.0	121	3.0	1.1	60.0	53.0	6
8788	Beef, raw, all grades, trimmed to 0" fat, sepa...	100.0	121	3.0	1.1	60.0	53.0	6

8789 rows × 74 columns

Dataset definition

In [100…

```
df.loc[240]
```

Out[100]:
```
name                                 McDONALD'S, BIG MAC
serving_size (g)                                   100.0
calories (kcal)                                      257
total_fat (g)                                       15.0
saturated_fat (g)                                    3.8
                                                    ...
fatty_acids_total_trans (mg)                        36.0
alcohol (g)                                          0.0
ash (g)                                             1.87
caffeine (mg)                                        0.0
theobromine (mg)                                     0.0
Name: 240, Length: 74, dtype: object
```

In [101…
```
df.loc[736]
```

Out[101]:
```
name                                 KEEBLER, Iced Oatmeal Cookies
serving_size (g)                                             100.0
calories (kcal)                                              467
total_fat (g)                                               18.0
saturated_fat (g)                                            5.8
                                                            ...
fatty_acids_total_trans (mg)                                 0.0
alcohol (g)                                                  0.0
ash (g)                                                      0.0
caffeine (mg)                                                0.0
theobromine (mg)                                             0.0
Name: 736, Length: 74, dtype: object
```

In [102…
```
df.head()
```

Out[102]:

	name	serving_size (g)	calories (kcal)	total_fat (g)	saturated_fat (g)	cholesterol (mg)	sodium (mg)	choline (mg)
0	Cornstarch	100.0	381	0.1	0.0	0.0	9.0	0.4
1	Nuts, pecans	100.0	691	72.0	6.2	0.0	0.0	40.5
2	Eggplant, raw	100.0	25	0.2	0.0	0.0	2.0	6.9
3	Teff, uncooked	100.0	367	2.4	0.4	0.0	12.0	13.1
4	Sherbet, orange	100.0	144	2.0	1.2	1.0	46.0	7.7

5 rows × 74 columns

Price df

In [103…
```
#abhishek's
price_df = pd.read_csv("Price_per_unit.csv", index_col=0)
```

In [104…
```
price_df
```

Out[104]:

	Ingredients	Prices	Weights	Price per unit
0	ANDREA'S	4.39	13.00	0.337692
1	APPLEBEE'S	0.00	0.00	10000.000000
2	ARBY'S	6.99	22.00	0.317727
3	ARCHWAY Home Style Cookies	5.49	9.25	0.593514
4	AUSTIN	0.00	0.00	10000.000000
...	...	...	...	...
589	Yellow rice with seasoning	8.49	2.00	4.245000
590	Yogurt	2.89	32.00	0.090313
591	Yogurt parfait	1.19	6.00	0.198333
592	Yokan	16.99	21.00	0.809048
593	Zwieback	0.00	0.00	10000.000000

1092 rows × 4 columns

In [105…

```
price_per_unit = {}
for index, row in price_df.iterrows():
    price_per_unit[row["Ingredients"]] = row["Price per unit"]
    if row['Weights'] == 1:
        price_per_unit[row["Ingredients"]] = row["Price per unit"] / 75
```

In [106…

```
for index, row in df.iterrows():
    curr = ''.join(row["name"].split(",")[:1])
    try:
        df.loc[index, "price_per_unit"] = price_per_unit[curr]
    except:
        df.loc[index, "price_per_unit"] = 10000
```

In [107…

```
# Get the serving size value
serving_size = df['serving_size (g)'].values[0]

# Divide all columns by the serving size value, except for the 'name' colum
for col in df.columns:
    if col != 'name':
        df[col] = df[col] / serving_size

# drop any columns that are all 0
df = df.loc[:, (df != 0).any(axis=0)]

# drop any rows that are all 0
df = df.loc[(df != 0).any(axis=1)]

df
```

Out[107]:

	name	serving_size (g)	calories (kcal)	total_fat (g)	saturated_fat (g)	cholesterol (mg)	sodium (mg)	cho (r
0	Cornstarch	1.0	3.81	0.001	0.000	0.00	0.09	0.(
1	Nuts, pecans	1.0	6.91	0.720	0.062	0.00	0.00	0..
2	Eggplant, raw	1.0	0.25	0.002	0.000	0.00	0.02	0.(
3	Teff, uncooked	1.0	3.67	0.024	0.004	0.00	0.12	0
4	Sherbet, orange	1.0	1.44	0.020	0.012	0.01	0.46	0.
...	...	...	...	...	...	...	...	
8784	Beef, raw, all grades, trimmed to 0" fat, sepa...	1.0	1.25	0.035	0.014	0.62	0.54	0.(
8785	Lamb, cooked, separable lean only, composite o...	1.0	2.06	0.089	0.039	1.09	0.50	0.(
8786	Lamb, raw, separable lean and fat, composite o...	1.0	2.77	0.230	0.120	0.78	0.39	0.(
8787	Beef, raw, all grades, trimmed to 0" fat, sepa...	1.0	1.21	0.030	0.011	0.60	0.53	0.(
8788	Beef, raw, all grades, trimmed to 0" fat, sepa...	1.0	1.21	0.030	0.011	0.60	0.53	0.(

8789 rows × 75 columns

In [108…

```
keys = ['calories (kcal)','total_fat (g)',
        'saturated_fat (g)', 'cholesterol (mg)', 'sodium (mg)', 'choline (mg)
        'folate (mcg)', 'folic_acid (mcg)', 'niacin (mg)',
        'pantothenic_acid (mg)', 'riboflavin (mg)', 'thiamin (mg)',
        'vitamin_a (IU)', 'vitamin_a_rae (mcg)', 'carotene_alpha (mcg)',
        'carotene_beta (mcg)', 'cryptoxanthin_beta (mcg)',
        'lutein_zeaxanthin (mcg)', 'vitamin_b12 (mg)', 'vitamin_b6 (mg)',
        'vitamin_c (mg)', 'vitamin_d (IU)', 'vitamin_e (mg)',
        'tocopherol_alpha (mg)', 'vitamin_k (mcg)', 'calcium (mg)',
        'copper (mg)', 'iron (mg)', 'magnesium (mg)', 'manganese (mg)',
        'phosphorous', 'potassium (mg)', 'selenium (mcg)', 'zinc (mg)',
        'protein (g)', 'alanine (g)', 'arginine (g)', 'aspartic_acid (g)',
        'cystine (g)', 'glutamic_acid (g)', 'glycine (g)', 'histidine (g)',
        'hydroxyproline (g)', 'isoleucine (g)', 'leucine (g)', 'lysine (g)',
        'methionine (g)', 'phenylalanine (g)', 'proline (g)', 'serine (g)',
```

```
        'threonine (g)', 'tryptophan (g)', 'tyrosine (g)', 'valine (g)',
        'carbohydrate (g)', 'fiber (g)', 'sugars (g)', 'fructose (g)',
        'galactose (g)', 'glucose (g)', 'lactose (g)', 'maltose (g)',
        'sucrose (g)', 'fat (g)', 'saturated_fatty_acids (g)',
        'monounsaturated_fatty_acids (g)', 'polyunsaturated_fatty_acids (g)',
        'fatty_acids_total_trans (mg)', 'alcohol (g)', 'ash (g)',
        'caffeine (mg)', 'theobromine (mg)']

# read the minimum and maximum values for each nutrient from excel file
df2 = pd.read_excel("Nutrient_values.xlsx")

# Cleaning the 'Nutrient' column by removing apostrophes and trailing commas
df2['Nutrient'] = df2['Nutrient'].str.replace("'", "").str.replace(",", "")

# Extracting two dictionaries: one for minimum values and one for maximum va
mini = df2.set_index('Nutrient')['Min_value'].to_dict()
maxi = df2.set_index('Nutrient')['Max_value'].to_dict()
```

Optimization Model

In [109…

```
import gurobipy as gb
import pandas as pd

# Assuming 'df' is a pandas DataFrame already defined with nutritional infor
# 'W' contains the decision variables representing ingredient weights

# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:
    if key in mini:
        model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.index)
    if key in maxi:
        model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.index)

#Constraint of the 1/5th (to remove any excesses of one particular ingredien
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()

# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
for i in W.keys():
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0x16f3e7b9
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 20 rows and 0 columns (presolve time = 7s) ...
Presolve removed 23 rows and 0 columns (presolve time = 16s) ...
Presolve removed 23 rows and 0 columns (presolve time = 72s) ...
Presolve removed 23 rows and 0 columns (presolve time = 76s) ...
Presolve removed 23 rows and 0 columns (presolve time = 84s) ...
Presolve removed 23 rows and 0 columns (presolve time = 86s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 87.43s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.29s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 80 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 140.69 seconds (432.64 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
     126    1.0835521e+00   0.000000e+00   0.000000e+00    150s

Solved in 126 iterations and 150.05 seconds (46.07 work units)
Optimal objective  1.083552094e+00
Optimal solution:
W[256] = 5.676798333138195
Ingredient: Fish oil, cod liver
W[285] = 0.5929759195615101
Ingredient: Acerola juice, raw
W[297] = 33.33275821212947
Ingredient: Onions, raw, welsh
W[356] = 17.873501390548142
Ingredient: Fish oil, herring
W[665] = 156.65345302937155
Ingredient: Yam, raw
W[696] = 34.704418721122856
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 39.03041462550613
Ingredient: Seeds, low-fat, sesame flour
W[2437] = 1.453650556034502
Ingredient: Nuts, blanched, hazelnuts or filberts
W[2857] = 9.573238858287318
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
W[3598] = 16.45002832836686
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
```

```
W[3810] = 156.65345302937158
Ingredient: Cardoon, without salt, drained, boiled, cooked
W[4291] = 35.29158066920942
Ingredient: Yam, without salt, or baked, drained, boiled, cooked
W[5058] = 61.8699630854659
Ingredient: Cereals, dry, 10 minute cooking, regular, CREAM OF WHEAT
W[5120] = 9.366579299337232
Ingredient: Cereals ready-to-eat, RALSTON Enriched Wheat Bran flakes
W[5312] = 10.399372835673548
Ingredient: USDA Commodity Food, low saturated fat, vegetable, oil
W[5334] = 134.17563260296305
Ingredient: Spaghetti, enriched (n x 6.25), dry, protein-fortified
W[5392] = 12.748092348793582
Ingredient: Wocas, yellow pond lily (Klamath), Oregon, dried seeds
W[5929] = 9.67365950923953
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
W[6220] = 0.5468103467066588
Ingredient: Beef, raw, liver, variety meats and by-products, imported, New
Zealand
W[6525] = 34.232059022512054
Ingredient: Whale, skin and subcutaneous fat (muktuk) (Alaska Native), bowh
ead
W[8028] = 2.9688244235188104
Ingredient: Margarine, CANOLA HARVEST Soft Spread (canola, palm and palm ke
rnel oils), tub, 80% fat
Objective Value (Minimum Price): 3.82212065153449
```

Optimal solution

In [110…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                Ingredient  Weight (g)
256                                  Fish oil, cod liver    5.676798
285                                   Acerola juice, raw    0.592976
297                                   Onions, raw, welsh   33.332758
356                                    Fish oil, herring   17.873501
665                                              Yam, raw  156.653453
696                          Tofu, dried-frozen (koyadofu)   34.704419
883                           Seeds, low-fat, sesame flour   39.030415
2437                Nuts, blanched, hazelnuts or filberts    1.453651
2857         KELLOGG'S, Original 3-Bean Chips, BEANATURAL    9.573239
3598   Sea lion, meat with fat (Alaska Native), Steller   16.450028
3810     Cardoon, without salt, drained, boiled, cooked  156.653453
4291  Yam, without salt, or baked, drained, boiled, ...   35.291581
5058  Cereals, dry, 10 minute cooking, regular, CREA...   61.869963
5120  Cereals ready-to-eat, RALSTON Enriched Wheat B...    9.366579
5312  USDA Commodity Food, low saturated fat, vegeta...   10.399373
5334  Spaghetti, enriched (n x 6.25), dry, protein-f...  134.175633
5392  Wocas, yellow pond lily (Klamath), Oregon, dri...   12.748092
5929  Tofu, prepared with calcium sulfate, dried-fro...     9.67366
6220  Beef, raw, liver, variety meats and by-product...     0.54681
6525  Whale, skin and subcutaneous fat (muktuk) (Ala...   34.232059
8028  Margarine, CANOLA HARVEST Soft Spread (canola,...    2.968824

Total cost of the diet: $3.82212065153449

Total weight of the diet: 783.267265146858 grams
```

Case studies

Case1: Iron-Deficieny Anaemia in pregnant women

In [111…

```python
# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:

    if key in mini:
        if key =='iron (mg)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
    if key in maxi:
        if key =='iron (mg)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in

#Constraint of the 1/5th(to remove any excesses of one particular ingredient
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()
```

```python
# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
for i in W.keys():
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0xfcc45001
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 18 rows and 0 columns (presolve time = 5s) ...
Presolve removed 23 rows and 0 columns (presolve time = 16s) ...
Presolve removed 23 rows and 0 columns (presolve time = 53s) ...
Presolve removed 23 rows and 0 columns (presolve time = 58s) ...
Presolve removed 23 rows and 0 columns (presolve time = 65s) ...
Presolve removed 23 rows and 0 columns (presolve time = 65s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 68.49s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.25s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 70 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 106.48 seconds (432.64 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
      83    1.2900128e+00   0.000000e+00   0.000000e+00    114s

Solved in 83 iterations and 114.43 seconds (45.99 work units)
Optimal objective  1.290012786e+00
Optimal solution:
W[256] = 5.6789739447503385
Ingredient: Fish oil, cod liver
W[285] = 1.531904705141629
Ingredient: Acerola juice, raw
W[297] = 24.34725942238764
Ingredient: Onions, raw, welsh
W[356] = 6.052562329218256
Ingredient: Fish oil, herring
W[665] = 105.41305639455075
Ingredient: Yam, raw
W[696] = 41.3093606976791
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 34.24091671120149
Ingredient: Seeds, low-fat, sesame flour
W[2085] = 136.67787884984597
Ingredient: Whale, raw (Alaska Native), meat, beluga
W[2437] = 23.848322695247756
Ingredient: Nuts, blanched, hazelnuts or filberts
W[2857] = 29.953680900285676
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
```

```
W[3162] = 2.0167006303514525
Ingredient: Cereals, dry, plain, original, MALT-O-MEAL
W[3598] = 18.06805756550674
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
W[3810] = 163.25201740430788
Ingredient: Cardoon, without salt, drained, boiled, cooked
W[5058] = 59.1077381992295
Ingredient: Cereals, dry, 10 minute cooking, regular, CREAM OF WHEAT
W[5120] = 9.30510305071482
Ingredient: Cereals ready-to-eat, RALSTON Enriched Wheat Bran flakes
W[5312] = 20.201245010202346
Ingredient: USDA Commodity Food, low saturated fat, vegetable, oil
W[5334] = 126.34459445753122
Ingredient: Spaghetti, enriched (n x 6.25), dry, protein-fortified
W[5392] = 2.5701793397024826
Ingredient: Wocas, yellow pond lily (Klamath), Oregon, dried seeds
W[5929] = 6.340534713684285
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
Objective Value (Minimum Price): 4.550389903280612
```

In [112…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                 Ingredient  Weight (g)
256                                 Fish oil, cod liver    5.678974
285                                  Acerola juice, raw    1.531905
297                                  Onions, raw, welsh   24.347259
356                                   Fish oil, herring    6.052562
665                                              Yam, raw  105.413056
696                        Tofu, dried-frozen (koyadofu)   41.309361
883                         Seeds, low-fat, sesame flour   34.240917
2085           Whale, raw (Alaska Native), meat, beluga  136.677879
2437              Nuts, blanched, hazelnuts or filberts   23.848323
2857       KELLOGG'S, Original 3-Bean Chips, BEANATURAL   29.953681
3162         Cereals, dry, plain, original, MALT-O-MEAL    2.016701
3598   Sea lion, meat with fat (Alaska Native), Steller   18.068058
3810     Cardoon, without salt, drained, boiled, cooked  163.252017
5058  Cereals, dry, 10 minute cooking, regular, CREA...   59.107738
5120  Cereals ready-to-eat, RALSTON Enriched Wheat B...    9.305103
5312  USDA Commodity Food, low saturated fat, vegeta...   20.201245
5334  Spaghetti, enriched (n x 6.25), dry, protein-f...  126.344594
5392  Wocas, yellow pond lily (Klamath), Oregon, dri...    2.570179
5929  Tofu, prepared with calcium sulfate, dried-fro...    6.340535

Total cost of the diet: $4.550389903280612

Total weight of the diet: 816.2600870215394 grams
```

Case 2 : Vitamin D deficiency

In [113…

```python
# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:

    if key in mini:
        if key =='vitamin_d (IU)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
    if key in maxi:
        if key =='vitamin_d (IU)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in

#Constraint of the 1/5th(to remove any excesses of one particular ingredient
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()

# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
for i in W.keys():
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
```

```
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0xca326616
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 0 rows and 0 columns (presolve time = 5s) ...
Presolve removed 23 rows and 0 columns (presolve time = 15s) ...
Presolve removed 23 rows and 0 columns (presolve time = 65s) ...
Presolve removed 23 rows and 0 columns (presolve time = 69s) ...
Presolve removed 23 rows and 0 columns (presolve time = 70s) ...
Presolve removed 23 rows and 0 columns (presolve time = 77s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 81.14s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.21s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 70 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 106.57 seconds (43.73 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
      96    3.4880629e+00   0.000000e+00   0.000000e+00    115s

Solved in 96 iterations and 115.09 seconds (45.81 work units)
Optimal objective  3.488062885e+00
Optimal solution:
W[180] = 7.871248285913017
Ingredient: Amaranth leaves, raw
W[256] = 9.056541354200336
Ingredient: Fish oil, cod liver
W[538] = 0.5311709446314284
Ingredient: Nuts, almonds
W[696] = 36.51282919213001
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 15.137086743751775
Ingredient: Seeds, low-fat, sesame flour
W[1270] = 30.877744182154398
Ingredient: Nuts, dried, butternuts
W[1291] = 207.86511664157234
Ingredient: Mushrooms, raw, maitake
W[2027] = 9.814194744634912
Ingredient: Fish, raw, Greenland, halibut
W[2857] = 95.65348427607054
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
W[3162] = 19.149785061644767
Ingredient: Cereals, dry, plain, original, MALT-O-MEAL
```

```
W[3372] = 10.966368226321217
Ingredient: Sea lion, liver (Alaska Native), Steller
W[3598] = 28.024233315196362
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
W[3621] = 10.808509526623311
Ingredient: Sweeteners, packets, EQUAL, aspartame, tabletop
W[4744] = 34.493254093683845
Ingredient: Sweetener, herbal extract powder from Stevia leaf
W[4930] = 11.564330024440991
Ingredient: Cereals ready-to-eat, Warm Cinnamon, KASHI HEART TO HEART
W[4947] = 207.86511664157234
Ingredient: Spaghetti, enriched (n x 6.25), cooked, protein-fortified
W[5058] = 43.38698464763886
Ingredient: Cereals, dry, 10 minute cooking, regular, CREAM OF WHEAT
W[5392] = 21.397535173146178
Ingredient: Wocas, yellow pond lily (Klamath), Oregon, dried seeds
W[5617] = 2.1870637547129483
Ingredient: Babyfood, organic, carrot and squash, apple, 2nd Foods, GERBER
W[5809] = 17.733616446714105
Ingredient: Pork, raw, frozen, ears, variety meats and by-products, fresh
W[5929] = 10.56425328953689
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
W[7436] = 207.86511664157237
Ingredient: Mushrooms, raw, exposed to ultraviolet light, or crimini, itali
an, brown
Objective Value (Minimum Price): 12.30378978557047
```

In [114…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                Ingredient  Weight (g)
180                                  Amaranth leaves, raw    7.871248
256                                   Fish oil, cod liver    9.056541
538                                         Nuts, almonds    0.531171
696                           Tofu, dried-frozen (koyadofu)   36.512829
883                            Seeds, low-fat, sesame flour   15.137087
1270                                Nuts, dried, butternuts   30.877744
1291                                Mushrooms, raw, maitake  207.865117
2027                           Fish, raw, Greenland, halibut    9.814195
2857         KELLOGG'S, Original 3-Bean Chips, BEANATURAL   95.653484
3162           Cereals, dry, plain, original, MALT-O-MEAL   19.149785
3372             Sea lion, liver (Alaska Native), Steller   10.966368
3598    Sea lion, meat with fat (Alaska Native), Steller   28.024233
3621     Sweeteners, packets, EQUAL, aspartame, tabletop    10.80851
4744  Sweetener, herbal extract powder from Stevia leaf   34.493254
4930  Cereals ready-to-eat, Warm Cinnamon, KASHI HEA...    11.56433
4947  Spaghetti, enriched (n x 6.25), cooked, protei...  207.865117
5058  Cereals, dry, 10 minute cooking, regular, CREA...   43.386985
5392  Wocas, yellow pond lily (Klamath), Oregon, dri...   21.397535
5617  Babyfood, organic, carrot and squash, apple, 2...    2.187064
5809  Pork, raw, frozen, ears, variety meats and by-...   17.733616
5929  Tofu, prepared with calcium sulfate, dried-fro...   10.564253
7436  Mushrooms, raw, exposed to ultraviolet light, ...  207.865117

Total cost of the diet: $12.30378978557047

Total weight of the diet: 1039.3255832078628 grams
```

Case 3: Protein heavy diet

In [115…

```python
# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:

    if key in mini:
        if key =='protein (g)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
    if key in maxi:
        if key =='protein (g)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in

#Constraint of the 1/5th(to remove any excesses of one particular ingredient
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()

# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
```

```
for i in W.keys():
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0xc98dff06
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 18 rows and 0 columns (presolve time = 5s) ...
Presolve removed 23 rows and 0 columns (presolve time = 13s) ...
Presolve removed 23 rows and 0 columns (presolve time = 45s) ...
Presolve removed 23 rows and 0 columns (presolve time = 49s) ...
Presolve removed 23 rows and 0 columns (presolve time = 51s) ...
Presolve removed 23 rows and 0 columns (presolve time = 55s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 58.45s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.82s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 70 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 81.67 seconds (43.73 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
      72    1.2299986e+00   0.000000e+00   0.000000e+00     90s

Solved in 72 iterations and 90.00 seconds (45.76 work units)
Optimal objective  1.229998619e+00
Optimal solution:
W[256] = 5.979222851877564
Ingredient: Fish oil, cod liver
W[285] = 3.04397874424587
Ingredient: Acerola juice, raw
W[297] = 22.6495112682897
Ingredient: Onions, raw, welsh
W[665] = 85.08871048395295
Ingredient: Yam, raw
W[696] = 14.439823436015232
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 34.2704666484676
Ingredient: Seeds, low-fat, sesame flour
W[2085] = 19.499197959655728
Ingredient: Whale, raw (Alaska Native), meat, beluga
W[2857] = 48.94759367122379
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
W[3598] = 5.159294568104607
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
W[3810] = 202.43682372423243
Ingredient: Cardoon, without salt, drained, boiled, cooked
```

```
W[4947] = 190.0256358938021
Ingredient: Spaghetti, enriched (n x 6.25), cooked, protein-fortified
W[5058] = 59.26577258423876
Ingredient: Cereals, dry, 10 minute cooking, regular, CREAM OF WHEAT
W[5120] = 0.602236177461909
Ingredient: Cereals ready-to-eat, RALSTON Enriched Wheat Bran flakes
W[5312] = 22.20829955661169
Ingredient: USDA Commodity Food, low saturated fat, vegetable, oil
W[5334] = 202.43682372423237
Ingredient: Spaghetti, enriched (n x 6.25), dry, protein-fortified
W[5392] = 33.630925579982595
Ingredient: Wocas, yellow pond lily (Klamath), Oregon, dried seeds
W[5929] = 7.653978150790162
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
W[6295] = 10.371225001335956
Ingredient: Pork, braised, cooked, pancreas, variety meats and by-products,
fresh
W[6525] = 44.47459859663973
Ingredient: Whale, skin and subcutaneous fat (muktuk) (Alaska Native), bowh
ead
Objective Value (Minimum Price): 4.338695988601508
```

In [116…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                Ingredient  Weight (g)
256                                  Fish oil, cod liver    5.979223
285                                   Acerola juice, raw    3.043979
297                                   Onions, raw, welsh   22.649511
665                                             Yam, raw    85.08871
696                         Tofu, dried-frozen (koyadofu)   14.439823
883                          Seeds, low-fat, sesame flour   34.270467
2085           Whale, raw (Alaska Native), meat, beluga   19.499198
2857       KELLOGG'S, Original 3-Bean Chips, BEANATURAL   48.947594
3598   Sea lion, meat with fat (Alaska Native), Steller    5.159295
3810     Cardoon, without salt, drained, boiled, cooked  202.436824
4947  Spaghetti, enriched (n x 6.25), cooked, protei...  190.025636
5058  Cereals, dry, 10 minute cooking, regular, CREA...   59.265773
5120  Cereals ready-to-eat, RALSTON Enriched Wheat B...    0.602236
5312  USDA Commodity Food, low saturated fat, vegeta...     22.2083
5334  Spaghetti, enriched (n x 6.25), dry, protein-f...  202.436824
5392  Wocas, yellow pond lily (Klamath), Oregon, dri...   33.630926
5929  Tofu, prepared with calcium sulfate, dried-fro...    7.653978
6295  Pork, braised, cooked, pancreas, variety meats...   10.371225
6525  Whale, skin and subcutaneous fat (muktuk) (Ala...   44.474599

Total cost of the diet: $4.338695988601508

Total weight of the diet: 1012.1841186211607 grams
```

Case 4: Calorie deficit diet

In [117…

```python
# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:

    if key in mini:
        if key =='calories (kcal)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
    if key in maxi:
        if key =='calories (kcal)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in

#Constraint of the 1/5th(to remove any excesses of one particular ingredient
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()

# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
for i in W.keys():
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
```

```
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0xa3bc8685
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 20 rows and 0 columns (presolve time = 6s) ...
Presolve removed 23 rows and 0 columns (presolve time = 12s) ...
Presolve removed 23 rows and 0 columns (presolve time = 51s) ...
Presolve removed 23 rows and 0 columns (presolve time = 56s) ...
Presolve removed 23 rows and 0 columns (presolve time = 60s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 62.84s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.26s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 70 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 83.50 seconds (43.73 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
     116    1.2072489e+00   0.000000e+00   0.000000e+00     92s

Solved in 116 iterations and 91.71 seconds (45.83 work units)
Optimal objective  1.207248947e+00
Optimal solution:
W[180] = 4.14951134940416
Ingredient: Amaranth leaves, raw
W[256] = 5.808210494294022
Ingredient: Fish oil, cod liver
W[285] = 0.8052958092447263
Ingredient: Acerola juice, raw
W[297] = 20.339132897977652
Ingredient: Onions, raw, welsh
W[665] = 136.90719803247654
Ingredient: Yam, raw
W[696] = 25.702935027415446
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 26.83000864425127
Ingredient: Seeds, low-fat, sesame flour
W[989] = 160.26446785460672
Ingredient: Turnips, unprepared, frozen
W[2437] = 50.3057162133433
Ingredient: Nuts, blanched, hazelnuts or filberts
W[2857] = 6.1608709353683375
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
W[3162] = 6.8155927110312025
```

```
Ingredient: Cereals, dry, plain, original, MALT-O-MEAL
W[3598] = 23.656707483178103
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
W[3810] = 160.2644678546069
Ingredient: Cardoon, without salt, drained, boiled, cooked
W[3875] = 31.917569135596985
Ingredient: Tofu, prepared with calcium sulfate, firm, raw
W[5120] = 5.236772260921432
Ingredient: Cereals ready-to-eat, RALSTON Enriched Wheat Bran flakes
W[5334] = 113.6148117897728
Ingredient: Spaghetti, enriched (n x 6.25), dry, protein-fortified
W[5744] = 1.669059068501886
Ingredient: Cereals ready-to-eat, Honey Toasted Oat, KASHI HEART TO HEART
W[5929] = 14.236109368179457
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
W[6157] = 0.8058596162455399
Ingredient: Cereals ready-to-eat, KELLOGG'S ALL-BRAN COMPLETE Wheat Flakes,
KELLOGG
W[6295] = 4.961046675298493
Ingredient: Pork, braised, cooked, pancreas, variety meats and by-products,
fresh
W[8491] = 0.8709960513198554
Ingredient: Gelatin desserts, vitamin C, sodium, potassium, added phosphoru
s, with aspartame, reduced calorie, dry mix
Objective Value (Minimum Price): 4.2584488171291
```

In [118…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                Ingredient  Weight (g)
180                                 Amaranth leaves, raw    4.149511
256                                  Fish oil, cod liver     5.80821
285                                   Acerola juice, raw    0.805296
297                                   Onions, raw, welsh   20.339133
665                                              Yam, raw  136.907198
696                         Tofu, dried-frozen (koyadofu)   25.702935
883                          Seeds, low-fat, sesame flour   26.830009
989                           Turnips, unprepared, frozen  160.264468
2437                Nuts, blanched, hazelnuts or filberts   50.305716
2857       KELLOGG'S, Original 3-Bean Chips, BEANATURAL    6.160871
3162         Cereals, dry, plain, original, MALT-O-MEAL    6.815593
3598   Sea lion, meat with fat (Alaska Native), Steller   23.656707
3810     Cardoon, without salt, drained, boiled, cooked  160.264468
3875     Tofu, prepared with calcium sulfate, firm, raw   31.917569
5120  Cereals ready-to-eat, RALSTON Enriched Wheat B...    5.236772
5334  Spaghetti, enriched (n x 6.25), dry, protein-f...  113.614812
5744  Cereals ready-to-eat, Honey Toasted Oat, KASHI...    1.669059
5929  Tofu, prepared with calcium sulfate, dried-fro...   14.236109
6157  Cereals ready-to-eat, KELLOGG'S ALL-BRAN COMPL...     0.80586
6295  Pork, braised, cooked, pancreas, variety meats...    4.961047
8491  Gelatin desserts, vitamin C, sodium, potassium...    0.870996

Total cost of the diet: $4.2584488171291

Total weight of the diet: 801.3223392730348 grams
```

Case 5: Scurvy patient

In [119…

```python
# Initialize the model
model = gb.Model()

# Create variables (change vtype to GRB.CONTINUOUS if fractional weights are
W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

# Set objective to minimize calories
obj = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjective(obj, gb.GRB.MINIMIZE)

# Assuming 'mini' and 'maxi' are defined correctly
# Add constraints for nutrient consumption
for key in keys:

    if key in mini:
        if key =='vitamin_c (mg)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
    if key in maxi:
        if key =='vitamin_c (mg)':
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in
        else:
            model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.in

#Constraint of the 1/5th(to remove any excesses of one particular ingredient
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()

# Print the optimal solution and the name of the ingredients used
print('Optimal solution:')
for i in W.keys():
```

```
    if W[i].X > 0:
        print(f'W[{i}] = {W[i].X}')
        print(f'Ingredient: {df.loc[i, "name"]}')

print(f'Objective Value (Minimum Price): {model.ObjVal * 100/28.3495}')
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0x89d878cb
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]
Presolve removed 20 rows and 0 columns (presolve time = 7s) ...
Presolve removed 24 rows and 0 columns (presolve time = 12s) ...
Presolve removed 24 rows and 0 columns (presolve time = 53s) ...
Presolve removed 24 rows and 0 columns (presolve time = 58s) ...
Presolve removed 24 rows and 0 columns (presolve time = 62s) ...
Presolve removed 24 rows and 0 columns (presolve time = 70s) ...
Presolve removed 76 rows and 0 columns
Presolve time: 73.69s
Presolved: 8857 rows, 8841 columns, 77600179 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 1.71s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 70 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 125.65 seconds (433.34 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
     114    1.0898859e+00   0.000000e+00   0.000000e+00    134s

Solved in 114 iterations and 134.38 seconds (46.76 work units)
Optimal objective  1.089885883e+00
Optimal solution:
W[256] = 5.660886859645235
Ingredient: Fish oil, cod liver
W[285] = 2.092729139178991
Ingredient: Acerola juice, raw
W[297] = 33.385053306452754
Ingredient: Onions, raw, welsh
W[356] = 17.721485863739183
Ingredient: Fish oil, herring
W[665] = 157.05463190570313
Ingredient: Yam, raw
W[696] = 34.33408181305669
Ingredient: Tofu, dried-frozen (koyadofu)
W[883] = 39.2459271554167
Ingredient: Seeds, low-fat, sesame flour
W[2437] = 0.7194305731409942
Ingredient: Nuts, blanched, hazelnuts or filberts
W[2857] = 10.100356228971815
Ingredient: KELLOGG'S, Original 3-Bean Chips, BEANATURAL
W[3598] = 16.383267151946193
Ingredient: Sea lion, meat with fat (Alaska Native), Steller
```

```
W[3810] = 157.05463190570313
Ingredient: Cardoon, without salt, drained, boiled, cooked
W[4291] = 33.82191403224329
Ingredient: Yam, without salt, or baked, drained, boiled, cooked
W[5058] = 60.276475289622326
Ingredient: Cereals, dry, 10 minute cooking, regular, CREAM OF WHEAT
W[5120] = 9.828108984638314
Ingredient: Cereals ready-to-eat, RALSTON Enriched Wheat Bran flakes
W[5312] = 10.347726124375374
Ingredient: USDA Commodity Food, low saturated fat, vegetable, oil
W[5334] = 134.71297151086378
Ingredient: Spaghetti, enriched (n x 6.25), dry, protein-fortified
W[5392] = 12.878290509694553
Ingredient: Wocas, yellow pond lily (Klamath), Oregon, dried seeds
W[5929] = 10.13465198401104
Ingredient: Tofu, prepared with calcium sulfate, dried-frozen (koyadofu)
W[6220] = 0.4338038474299795
Ingredient: Beef, raw, liver, variety meats and by-products, imported, New
Zealand
W[6525] = 37.169527946339656
Ingredient: Whale, skin and subcutaneous fat (muktuk) (Alaska Native), bowh
ead
W[8028] = 1.9172073963421377
Ingredient: Margarine, CANOLA HARVEST Soft Spread (canola, palm and palm ke
rnel oils), tub, 80% fat
Objective Value (Minimum Price): 3.844462452287473
```

In [120…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution)

# Print the total cost of the diet
print(f'\nTotal cost of the diet: ${model.ObjVal * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

```
                                                Ingredient  Weight (g)
256                                  Fish oil, cod liver    5.660887
285                                   Acerola juice, raw    2.092729
297                                   Onions, raw, welsh   33.385053
356                                    Fish oil, herring   17.721486
665                                              Yam, raw  157.054632
696                          Tofu, dried-frozen (koyadofu)   34.334082
883                           Seeds, low-fat, sesame flour   39.245927
2437               Nuts, blanched, hazelnuts or filberts    0.719431
2857        KELLOGG'S, Original 3-Bean Chips, BEANATURAL   10.100356
3598   Sea lion, meat with fat (Alaska Native), Steller   16.383267
3810     Cardoon, without salt, drained, boiled, cooked  157.054632
4291  Yam, without salt, or baked, drained, boiled, ...   33.821914
5058  Cereals, dry, 10 minute cooking, regular, CREA...   60.276475
5120  Cereals ready-to-eat, RALSTON Enriched Wheat B...    9.828109
5312  USDA Commodity Food, low saturated fat, vegeta...   10.347726
5334  Spaghetti, enriched (n x 6.25), dry, protein-f...  134.712972
5392  Wocas, yellow pond lily (Klamath), Oregon, dri...   12.878291
5929  Tofu, prepared with calcium sulfate, dried-fro...   10.134652
6220  Beef, raw, liver, variety meats and by-product...    0.433804
6525  Whale, skin and subcutaneous fat (muktuk) (Ala...   37.169528
8028  Margarine, CANOLA HARVEST Soft Spread (canola,...    1.917207

Total cost of the diet: $3.844462452287473

Total weight of the diet: 785.2731595285155 grams
```

Case 6: God's diet (Maxed-out Diet)

Use of the multi-objective gurobi environment

In [121…

```python
model = gb.Model("Multi_Objective")

W = model.addVars(df.index, vtype=gb.GRB.CONTINUOUS, name='W')

#Setting the maximization of the nutrient value as the primary objective fur
obj1 = -1*gb.quicksum(gb.quicksum(W[i] * df.loc[i, key] for key in keys) for
model.setObjectiveN(obj1, 0, 1)

#Setting the minimization of the price as the secondary objective function
obj2 = gb.quicksum(W[i] * df.loc[i, 'price_per_unit'] for i in df.index)
model.setObjectiveN(obj2, 1, 0)

model.ModelSense = gb.GRB.MINIMIZE

for key in keys:
    if key in mini:
        model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.index)
    if key in maxi:
        model.addConstr(gb.quicksum(W[i] * df.loc[i, key] for i in df.index)

#Constraint of the 1/5th (to remove any excesses of one particular ingredien
model.addConstrs( W[i]<= (gb.quicksum(W[j] for j in df.index))/5 for i in df

# Optimize the model
model.optimize()
```

```
Gurobi Optimizer version 10.0.3 build v10.0.3rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 8933 rows, 8789 columns and 77977215 nonzeros
Model fingerprint: 0x6edc0741
Variable types: 8789 continuous, 0 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e-05, 1e+03]
  Objective range  [3e-05, 1e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-01, 3e+05]

------------------------------------------------------------------------------
Multi-objectives: starting optimization with 2 objectives ...
------------------------------------------------------------------------------

Multi-objectives: applying initial presolve ...
------------------------------------------------------------------------------

Presolve removed 20 rows and 0 columns (presolve time = 5s) ...
Presolve removed 23 rows and 0 columns (presolve time = 12s) ...
Presolve removed 23 rows and 0 columns (presolve time = 52s) ...
Presolve removed 23 rows and 0 columns (presolve time = 56s) ...
Presolve removed 23 rows and 0 columns (presolve time = 63s) ...
Presolve removed 23 rows and 0 columns
Presolved: 8910 rows and 8789 columns
------------------------------------------------------------------------------

Multi-objectives: optimize objective 1 () ...
------------------------------------------------------------------------------

Presolve removed 0 rows and 0 columns (presolve time = 5s) ...
Presolve removed 0 rows and 0 columns (presolve time = 14s) ...
Presolve removed 0 rows and 0 columns (presolve time = 37s) ...
Presolve removed 0 rows and 0 columns (presolve time = 40s) ...
Presolve removed 53 rows and 0 columns
Presolve time: 111.75s
Presolved: 8857 rows, 8842 columns, 77600180 nonzeros

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Ordering time: 2.14s

Barrier statistics:
 AA' NZ     : 3.922e+07
 Factor NZ  : 3.923e+07 (roughly 300 MB of memory)
 Factor Ops : 2.316e+11 (roughly 100 seconds per iteration)
 Threads    : 3

Barrier performed 0 iterations in 156.77 seconds (456.65 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
     178   -5.6391048e+04   0.000000e+00   0.000000e+00    166s

Solved in 178 iterations and 166.02 seconds (70.30 work units)
Optimal objective -5.639104827e+04
------------------------------------------------------------------------------
```

```
Multi-objectives: optimize objective 2 () ...
------------------------------------------------------------------------------

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.6401431e+04   0.000000e+00   3.179171e-01    181s
       2    2.4927089e+04   0.000000e+00   0.000000e+00    181s

Solved in 2 iterations and 181.18 seconds (71.08 work units)
Optimal objective  2.492708924e+04

------------------------------------------------------------------------------
Multi-objectives: solved in 182.16 seconds (71.08 work units), solution cou
nt 2
```

In [122…

```python
# Print the optimal solution and the name of the ingredients used in tabular

# Create a list of the ingredients used
ingredients = []
for i in W.keys():
    if W[i].X > 0:
        ingredients.append(df.loc[i, "name"])

# Create a DataFrame with the optimal solution
solution = pd.DataFrame(columns=['Ingredient', 'Weight (g)'])

for i in W.keys():
    if W[i].X > 0:
        solution.loc[i, 'Ingredient'] = df.loc[i, 'name']
        solution.loc[i, 'Weight (g)'] = W[i].X

# Print the DataFrame
print(solution[["Ingredient", "Weight (g)"]])

# Print the total cost of the diet
obj_val = gb.quicksum(W[i].X * df.loc[i, 'price_per_unit'] for i in df.index
print(f'\nTotal cost of the diet: ${obj_val * 100/28.3495}')

# Print the total weight of the diet
print(f'\nTotal weight of the diet: {sum(solution["Weight (g)"])} grams')
```

	Ingredient	Weight (g)
67	Oil, soybean lecithin	7.762224
180	Amaranth leaves, raw	1.286512
256	Fish oil, cod liver	1.221551
450	Oil, wheat germ	4.36787
711	Pumpkin, without salt, canned	52.305005
1378	Soy protein isolate, potassium type	19.088659
1618	Gelatins, unsweetened, dry powder	40.387709
1761	Cereals, dry, unenriched, farina	69.155478
2057	SMART SOUP, Vietnamese Carrot Lemongrass	154.707742
2140	KASHI, unprepared, 7 Whole Grain, Pilaf	86.243194
2601	Beverages, sugar free, Energy Drink	402.474342
2775	KASHI, Frozen Entree, Chicken Pasta Pomodoro	113.972411
2828	Beverages, decaffeinated, brewed, green, tea	87684.987617
3372	Sea lion, liver (Alaska Native), Steller	42.70884
3598	Sea lion, meat with fat (Alaska Native), Steller	58.387807
4285	Bamboo shoots, without salt, drained, boiled, ...	460.13465
4480	Beverages, Glacial Natural spring water, ICELA...	87956.50965
4583	Beverages, brewed, other than chamomile, herb,...	20239.6774
4668	Beverages, DANNON, non-carbonated, bottled, water	87956.50965
4671	Candies, dietetic or low calorie (sorbitol), hard	151.289218
4744	Sweetener, herbal extract powder from Stevia leaf	596.310724
4899	Babyfood, without added fluoride., GERBER, bot...	87956.50965
5617	Babyfood, organic, carrot and squash, apple, 2...	58.823529
5767	Beverages, fortified with vitamin C, Apple jui...	406.436241
6209	Alcoholic beverage, all (gin, rum, vodka, whis...	70.528967
6272	Beverages, diet, peach, ready to drink, black ...	3390.102631
7403	Beverages, non-carbonated, bottled, water, DAS...	61583.347213
7436	Mushrooms, raw, exposed to ultraviolet light, ...	161.693379
7774	Oil, woks and light frying, principal uses sal...	41.822239
7839	Beverages, fortified with vitamin C, unsweeten...	13.796147

Total cost of the diet: $87927.79144622775

Total weight of the diet: 439782.5482489649 grams