EEP153_Project 2_Draft_Atwater

March 4, 2022

1 EEP 153 Project 2: Group Atwater

1.0.1 Minimum Cost Diet:

Topic: Cost comparison of a minimized vegetarian diet for Berkeley residents across 5 different local grocery stores (Safeway, Trader Joe's, Amazon Fresh, Berkeley Bowl, Sprout's).

Objectives:

1. Explore different grocery store options to maintain a cost-minimized vegetarian diet in the city of Berkeley using a linear programming model based on the Stigler Diet Problem.
2. Create and evaluate vegetarian recipes that meet the minimum nutrient requirements for different sex and age groups
3. Utilize multiple data visualizations to compare and contrast the cost efficiency of a vegetarian diet across 5 different grocery stores in Berkeley

1.1 Table of contents

1.2 Import All Data Libraries

[1]:
```
!pip install -r requirements.txt
!pip install eep153_tools

import pandas as pd
import pandas as pd
import numpy as np
import fooddatacentral as fdc
from  scipy.optimize import linprog as lp

import warnings
```

```python
import ipywidgets
from ipywidgets import interactive, fixed, interact, Dropdown

import plotly.offline as py
import plotly.graph_objs as go
import cufflinks as cf
cf.go_offline()

from eep153_tools.sheets import read_sheets

#personal apikey from FDC
apikey = "kY45fKdbAFHas9GpxBKlDyEYbwvfC00z17oKd3ba"
```

```
Requirement already satisfied: numpy>=1.20.3 in /opt/conda/lib/python3.9/site-
packages (from -r requirements.txt (line 4)) (1.21.5)
Requirement already satisfied: pandas>=1.2.5 in /opt/conda/lib/python3.9/site-
packages (from -r requirements.txt (line 7)) (1.3.5)
Collecting pint>=0.18
  Using cached Pint-0.18-py2.py3-none-any.whl (209 kB)
Requirement already satisfied: requests>=2.26.0 in
/opt/conda/lib/python3.9/site-packages (from -r requirements.txt (line 13))
(2.26.0)
Collecting eep153_tools
  Using cached eep153_tools-0.11-py2.py3-none-any.whl (4.4 kB)
Processing /home/jovyan/.cache/pip/wheels/20/7e/30/7d702acd6a1e89911301cd9dbf9cb
9870ca80c0e64bc2cde23/gnupg-2.3.1-py3-none-any.whl
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/conda/lib/python3.9/site-packages (from pandas>=1.2.5->-r requirements.txt
(line 7)) (2.8.0)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.9/site-
packages (from pandas>=1.2.5->-r requirements.txt (line 7)) (2021.1)
Requirement already satisfied: packaging in /opt/conda/lib/python3.9/site-
packages (from pint>=0.18->-r requirements.txt (line 10)) (21.3)
Requirement already satisfied: idna<4,>=2.5; python_version >= "3" in
/opt/conda/lib/python3.9/site-packages (from requests>=2.26.0->-r
requirements.txt (line 13)) (2.8)
Requirement already satisfied: charset-normalizer~=2.0.0; python_version >= "3"
in /opt/conda/lib/python3.9/site-packages (from requests>=2.26.0->-r
requirements.txt (line 13)) (2.0.0)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.9/site-packages (from requests>=2.26.0->-r
requirements.txt (line 13)) (2019.11.28)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.9/site-packages (from requests>=2.26.0->-r
requirements.txt (line 13)) (1.25.7)
Requirement already satisfied: psutil>=1.2.1 in /opt/conda/lib/python3.9/site-
packages (from gnupg->-r requirements.txt (line 17)) (5.9.0)
```

```
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.9/site-
packages (from python-dateutil>=2.7.3->pandas>=1.2.5->-r requirements.txt (line
7)) (1.16.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.9/site-packages (from packaging->pint>=0.18->-r
requirements.txt (line 10)) (3.0.7)
Installing collected packages: pint, eep153-tools, gnupg
  Attempting uninstall: pint
    Found existing installation: Pint 0.17
    Uninstalling Pint-0.17:
      Successfully uninstalled Pint-0.17
Successfully installed eep153-tools-0.11 gnupg-2.3.1 pint-0.18
Requirement already satisfied: eep153_tools in /opt/conda/lib/python3.9/site-
packages (0.11)
```

```
/opt/conda/lib/python3.9/site-packages/geopandas/_compat.py:111: UserWarning:

The Shapely GEOS version (3.10.2-CAPI-1.16.0) is incompatible with the GEOS
version PyGEOS was compiled with (3.10.1-CAPI-1.16.0). Conversions between both
will be slow.
```

1.3 [A] Description of Population of Interest

We are focusing on the **vegetarian population in Berkeley, California**.

In 2021, the city of Berkeley passed a resolution to slash the amount of animal products the city purchases by 50% by 2024. The resolution also adopted a long-term goal of phasing out all purchases of animal products and replacing them with plant-based foods.

With an increased demand in vegetarian and vegan products, our project will present minimum cost vegetarian recipes with ingredients purchased from local Berkeley grocery stores, while still consuming the minimum nutrients required for the appropriate age groups.

1.4 [A] Dietary Reference Intakes

A function that takes as arguments the characteristics of a person (age, sex) and returns a `pandas.Series` of Dietary Reference Intakes (DRI's) or "Recommended Daily Allowances" (RDA) of a variety of nutrients appropriate for your population of interest.

Input Parameters:

- **sex**: a str ('m', 'male', 'f', or 'female')
- **age**: an integer

[2]:
```
#function that takes in the age and sex of an individual and returns the␣
↪necessary nutritional intake for survival.
#sex must be either male or female or M or F
#age must be an integer
def necessary_nutrients(age, sex):
    bmin = pd.read_csv('./diet_minimums.csv').set_index('Nutrition').iloc[:,2:]
```

```python
    headers = bmin.columns.values
    sex = sex.lower()
    male = lambda x:x if 'M' in x or 'C' in x else None
    female = lambda x:x if 'F' in x or 'C' in x else None
    if sex == 'male' or sex == 'm':
        filtered = [male(x) for x in headers]
        cleaned = [x for x in filtered if x is not None]
    elif sex == 'female' or sex == 'f':
        filtered = [female(x) for x in headers]
        cleaned = [x for x in filtered if x is not None]
    if age < 4:
        return bmin[cleaned[0]]
    elif age < 9:
        return bmin[cleaned[1]]
    elif age < 14:
        return bmin[cleaned[2]]
    elif age < 19:
        return bmin[cleaned[3]]
    elif age < 31:
        return bmin[cleaned[4]]
    else:
        return bmin[cleaned[5]]
```

Example:

[3]:
```python
necessary_nutrients(age = 22, sex = 'male')
```

[3]:
```
Nutrition
Energy                              2400.0
Protein                               56.0
Fiber, total dietary                  33.6
Folate, DFE                          400.0
Calcium, Ca                         1000.0
Carbohydrate, by difference          130.0
Iron, Fe                               8.0
Magnesium, Mg                        400.0
Niacin                                16.0
Phosphorus, P                        700.0
Potassium, K                        4700.0
Riboflavin                             1.3
Thiamin                                1.2
Vitamin A, RAE                       900.0
Vitamin B-12                           2.4
Vitamin B-6                            1.3
Vitamin C, total ascorbic acid        90.0
Vitamin E (alpha-tocopherol)          15.0
Vitamin K (phylloquinone)            120.0
```

```
Zinc, Zn                                     11.0
Name: M 19-30, dtype: float64
```

[4]:
```
DRI_url = "https://docs.google.com/spreadsheets/d/
↪1y95IsQ4HKspPW3HHDtH7QMtlDA66IUsCHJLutVL-MMc/"

DRIs = read_sheets(DRI_url)

# Define *minimums*
diet_min = DRIs['diet_minimums'].set_index('Nutrition')

# Define *maximums*
diet_max = DRIs['diet_maximums'].set_index('Nutrition')
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

1.5 [A] Google Sheet on Prices for Different Foods

We have a basket of 45 different vegetarian food with prices from 5 different grocery stores around Berkeley (**Safeway, Trader Joe's, Amazon Fresh, Berkeley Bowl, Sprout's**)

[5]:
```
food_list = "https://docs.google.com/spreadsheets/d/
↪1qAbI2YBx-AV7IO_isbAJlCRNP-w40WxhcgfLH2USoYs/edit?usp=sharing"
```

To access the different price breakdown from different stores, use the `food_df` function below that creates a DataFrame.

Input Parameter:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")

[6]:
```
def food_df(store):
    df = read_sheets(food_list)[f"{store}"]
    return df
```

IMPORTANT: if you run into an API Error (for any of the function below), it is due to the stability of the server. Try rerunning the "import data libraies" cell and the function cell again; try reinitializing student.json with the "noodle octopus" passphrase; try restarting the kernel; or trying to replace the API key with your own. Unfortunately, it is possible for the problem to persist :(

[7]:
```
#use the interact function to explore food info for different stores
interact(food_df, store=["Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley␣
↪Bowl", "Sprout's"])
```

```
interactive(children=(Dropdown(description='store', options=('Safeway', "Trader␣
↪Joe's", 'Amazon Fresh', 'Berke…
```

[7]: <function __main__.food_df(store)>

1.5.1 Unit Conversion and Price per Unit

In order to compare prices, the `unit_convert` function takes in a store name, converts all product into FDC unit, and returns a list of prices per unit for each food.

Input Parameter:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")

[8]:
```
def unit_convert(store):
    df = food_df(store)

    # Convert food quantities to FDC units
    df['FDC Quantity'] = df[['Quantity','Units']].T.apply(lambda x : fdc.
 ↪units(x['Quantity'],x['Units']))

    # Now divide price by the FDC Quantity to get, e.g., price per hectoliter
    df['FDC Price'] = df['Price']/df['FDC Quantity']

    df.dropna(how='any') # Drop food with any missing data

    # To use minimum price observed
    Prices = df.groupby('Food')['FDC Price'].min()

    return Prices
```

[9]:
```
#use the interact function to explore food info for different stores
interact(unit_convert, store=["Safeway", "Trader Joe's", "Amazon Fresh",␣
 ↪"Berkeley Bowl", "Sprout's"])
```

```
interactive(children=(Dropdown(description='store', options=('Safeway', "Trader␣
 ↪Joe's", 'Amazon Fresh', 'Berke…
```

[9]: <function __main__.unit_convert(store)>

[10]:
```
safeway_p = unit_convert("Safeway")
TJ_p = unit_convert("Trader Joe's")
AF_p = unit_convert("Amazon Fresh")
BB_p = unit_convert("Berkeley Bowl")
sprouts_p = unit_convert("Sprout's")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
Key available for students@eep153.iam.gserviceaccount.com.
Key available for students@eep153.iam.gserviceaccount.com.
Key available for students@eep153.iam.gserviceaccount.com.
Key available for students@eep153.iam.gserviceaccount.com.
```

1.6 [A] Nutritional Content of Different Foods

The 'nutrient' function takes in a store name and returns a DataFrame with the nutritional content of **45 different food** according to their FDC ID.

Input Parameter:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")

[11]:
```
def nutrient(store):
    df = food_df(store)
    D = {}
    count = 0
    for food in  df.Food.tolist():
        try:
            FDC = df.loc[df.Food==food,:].FDC[count]
            count+=1
            D[food] = fdc.nutrients(apikey,FDC).Quantity
        except AttributeError:
            warnings.warn("Couldn't find FDC Code %s for food %s." % (food,FDC))

    D = pd.DataFrame(D,dtype=float)

    return D
```

Generate the nutrient df for the 5 different stores:

(These 5 cells do take a significantly longer time to run due to the large basket of goods)

[12]:
```
safeway_n = nutrient("Safeway")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

[13]:
```
TJ_n = nutrient("Trader Joe's")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

[14]:
```
AF_n = nutrient("Amazon Fresh")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

[15]:
```
BB_n = nutrient("Berkeley Bowl")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

[16]:
```
sprouts_n = nutrient("Sprout's")
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

1.7 [A] Solution to the Minimum-Cost-Diet Problem

The `solve_subsistence_problem` function generates a solution for Stigler's Subsistence Cost Problem using the linear programming model

$$\min_x p'x$$

such that

$$Ax \geq b$$

Input Parameters:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")
- **FoodNutrients** : A pd.DataFrame for the corresponding store with rows corresponding to foods, columns to nutrients.
- **Prices**: A pd.Series of prices for different foods
- **diet_min**: A pd.Series of DRIs, with index corresponding to columns of FoodNutrients,describing minimum intakes.
- **diet_max**: A pd.Series of DRIs, with index corresponding to columns of FoodNutrients, describing maximum intakes.
- **max_weight**: Optional argument; maximum weight (in hectograms) allowed for diet.
- **tol**: Solution values smaller than this in absolute value treated as zeros; default = 1e-6.

[17]:
```python
def solve_subsistence_problem(store, FoodNutrients, Prices, diet_min, diet_max,␣
↪max_weight=None,tol=1e-6):
    p = Prices.apply(lambda x:x.magnitude).dropna()

    # Compile list that we have both prices and nutritional info for; drop if␣
↪either missing
    use = p.index.intersection(FoodNutrients.columns)
    p = p[use]

    # Drop nutritional information for foods we don't know the price of,
    # and replace missing nutrients with zeros.
    Aall = FoodNutrients[p.index].fillna(0)

    # Drop rows of A that we don't have constraints for.
    Amin = Aall.loc[Aall.index.intersection(diet_min.index)]

    Amax = Aall.loc[Aall.index.intersection(diet_max.index)]

    # Minimum requirements involve multiplying constraint by -1 to make <=.
    A = pd.concat([Amin,
                   -Amax])

    b = pd.concat([diet_min,
                   -diet_max]) # Note sign change for max constraints

    # Make sure order of p, A, b are consistent
```

```python
    A = A.reindex(p.index,axis=1)
    A = A.reindex(b.index,axis=0)

    if max_weight is not None:
        # Add up weights of foods consumed
        A.loc['Hectograms'] = -1
        b.loc['Hectograms'] = -max_weight

    # Now solve problem!  (Note that the linear program solver we'll use assumes
    # "less-than-or-equal" constraints.  We can switch back and forth by
    # multiplying $A$ and $b$ by $-1$.)

    result = lp(p, -A, -b, method='interior-point')

    result.A = A
    result.b = b

    if result.success:
        result.diet = pd.Series(result.x,index=p.index)
    else: # No feasible solution?
        warnings.warn(result.message)
        result.diet = pd.Series(result.x,index=p.index)*np.nan

    return result
```

[18]:
```python
#some helper functions to shorten the code below
def match_n(store):
    if store == "Safeway":
        FoodNutrients = safeway_n
    elif store == "Trader Joe's":
        FoodNutrients = TJ_n
    elif store == "Amazon Fresh":
        FoodNutrients = AF_n
    elif store == "Berkeley Bowl":
        FoodNutrients = BB_n
    elif store == "Sprout's":
        FoodNutrients = sprouts_n
    return FoodNutrients
def match_p(store):
    if store == "Safeway":
        Prices = safeway_p
    elif store == "Trader Joe's":
        Prices = TJ_p
    elif store == "Amazon Fresh":
        Prices = AF_p
    elif store == "Berkeley Bowl":
        Prices = BB_p
```

```
    elif store == "Sprout's":
        Prices = sprouts_p
    return Prices
```

The `cheapest_diet` generate a solution for a specfic store.

Input Parameters:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")
- **sex**: a str ("M" or "F"
- **age**: a str ("4-8", "9-13", "14-18","19-30","31-50", or "51+")

[19]:
```
def cheapest_diet(store, sex, age):

    group = f'{sex } {age}'
    tol=1e-6

    FoodNutrients = match_n(store)
    Prices = match_p(store)

    result = solve_subsistence_problem(store, FoodNutrients, Prices,␣
↪diet_min[group], diet_max[group], tol=1e-6)

    group = "Vegetarian " + group

    print("Cost of diet from %a for %s is $%4.2f per day.\n" % (store,␣
↪group,result.fun))

    # Put back into nice series
    diet = result.diet

    print("\nDiet (in 100s of grams or milliliters):")
    print(diet[diet >= tol])  # Drop items with quantities less than precision␣
↪of calculation.
    print()

    tab = pd.DataFrame({"Outcome":np.abs(result.A).dot(diet),"Recommendation":
↪np.abs(result.b)})
    print("\nWith the following nutritional outcomes of interest:")
    print(tab)
    print()

    print("\nConstraining nutrients are:")
    excess = tab.diff(axis=1).iloc[:,1]
    print(excess.loc[np.abs(excess) < tol*100].index.tolist())
```

[20]:
```python
# Sample solution from Safeway for a male ages 19-30
cheapest_diet("Amazon Fresh", "M", "19-30")
```

```
Cost of diet from 'Amazon Fresh' for Vegetarian M 19-30 is $3.70 per day.


Diet (in 100s of grams or milliliters):
Almond Milk (Unsweetened)    3.125955
Black Beans (Canned)         5.273971
Carrots                      0.389967
Corn (canned)                7.806742
Kale                         0.237089
Potatoes (Russet)            2.500766
Whole Milk                   4.444444
dtype: float64


With the following nutritional outcomes of interest:
                                    Outcome  Recommendation
Nutrition
Energy                          4922.200477          2400.0
Protein                           97.000213            56.0
Fiber, total dietary              79.754872            33.6
Folate, DFE                      473.522411           400.0
Calcium, Ca                     1537.948918          1000.0
Carbohydrate, by difference      403.901699           130.0
Iron, Fe                          20.956168             8.0
Magnesium, Mg                    588.745946           400.0
Niacin                            24.779129            16.0
Phosphorus, P                   2186.444507           700.0
Potassium, K                    6674.992756          4700.0
Riboflavin                         2.083363             1.3
Thiamin                            1.983076             1.2
Vitamin A, RAE                   900.000231           900.0
Vitamin B-12                       2.400000             2.4
Vitamin B-6                        2.040287             1.3
Vitamin C, total ascorbic acid    90.000001            90.0
Vitamin E (alpha-tocopherol)      15.000000            15.0
Vitamin K (phylloquinone)        120.000001           120.0
Zinc, Zn                          11.000000            11.0
Sodium, Na                      2299.999887          2300.0


Constraining nutrients are:
['Vitamin B-12', 'Vitamin C, total ascorbic acid', 'Vitamin E (alpha-
tocopherol)', 'Vitamin K (phylloquinone)', 'Zinc, Zn']
```

[21]:
```
#Interactive solutio with respect to different store, sex and age
interact(cheapest_diet,
         store = ["Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl",␣
↪"Sprout's"],
         sex = ["M","F"],
         age = ["4-8", "9-13", "14-18","19-30","31-50","51+"])
```

```
interactive(children=(Dropdown(description='store', options=('Safeway', "Trader␣
↪Joe's", 'Amazon Fresh', 'Berke…
```

[21]:
```
<function __main__.cheapest_diet(store, sex, age)>
```

1.7.1 Final Result:

Rank of stores according to prices (ascending order):

1. Amazon Fresh
2. Safeway
3. Sprout's
4. Berkeley Bowl
5. Trader Joe's

Amazon Fresh offers the cheapest solutions across all sex and age group

- 3.7 dollar per day for a male aged 19-30
- 3 dollar per day for a female aged 19-30
- **However, it is critical to address that Amazon Fresh requires a delivery fee, ranging from \$2.99 to \$5; the order need to be up to \$35 for free delivery. Tips might also apply**

1.8 [B] Is our solution edible?

1.9 [B] Meal Reviews

1.10 [C] Sensitivity of Solution

1.10.1 1. Effect of Food Prices on Total Diet Cost

The `price_cost` function create a graph demonstrating **the relationship between the changes in prices for each food and the total diet cost** for the specified store and sex-age group

Input Parameters:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")
- **sex**: a str ("M" or "F"
- **age**: a str ("4-8", "9-13", "14-18","19-30","31-50", or "51+")

[22]:
```
def price_cost(store, sex, age):
    group = f'{sex } {age}'
    FoodNutrients = match_n(store)
    Prices = match_p(store)
```

```
    tol=1e-6

    scale = [.5,.6,.7,.8,.9,1.,1.1,1.2,1.3,1.4,1.5]
    cost0 = solve_subsistence_problem(store, FoodNutrients, Prices,␣
↪diet_min[group], diet_max[group], tol=1e-6).fun


    Price_response={}
    for s in scale:
        cost = {}
        for i,p in enumerate(Prices):
            my_p = Prices.copy()
            my_p[i] = p*s
            result = solve_subsistence_problem(store,␣
↪FoodNutrients,my_p,diet_min[group],diet_max[group],tol=tol)
            cost[Prices.index[i]] = np.log(result.fun/cost0)
        Price_response[np.log(s)] = cost

    Price_response = pd.DataFrame(Price_response).T
    return Price_response.iplot(xTitle='change in log price',
                                yTitle='change in log cost',
                                title=f"Change in Food Price vs Change in Diet␣
↪Cost ({store}, {group})")
```

[23]:
```
#interactive plot showing the effect of prices on total cost from different␣
↪stores for different sex-age groups
#the plots take a bit of time to load
interact(price_cost,
         store = ["Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl",␣
↪"Sprout's"],
         sex = ["M","F"],
         age = ["4-8", "9-13", "14-18","19-30","31-50","51+"])
```

```
interactive(children=(Dropdown(description='store', options=('Safeway', "Trader␣
↪Joe's", 'Amazon Fresh', 'Berke…
```

[23]:
```
<function __main__.price_cost(store, sex, age)>
```

1.10.2 2. Effect of Price of One Good on Diet Composition

The `price_quantity` function create a graph demonstrating **the relationship between the changes in price of 1 particular good and quantity of other goods consumed** for the specified store and sex-age group.

Input Parameters:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")

- **sex**: a str ("M" or "F"
- **age**: a str ("4-8", "9-13", "14-18","19-30","31-50", or "51+")

[25]:
```
def price_quantity(store, sex, age, good):
    group = f'{sex } {age}'
    FoodNutrients = match_n(store)
    Prices = match_p(store)
    tol=1e-6
    ReferenceGood = good

    scale = [0.5,0.75,0.9,1.,1.1,1.2,1.3,1.4,1.5,2,4]
    cost0 = solve_subsistence_problem(store, FoodNutrients, Prices,␣
 ↪diet_min[group], diet_max[group], tol=1e-6).fun

    my_p = Prices.copy()

    diet = {}
    for s in scale:
        my_p[ReferenceGood] = Prices[ReferenceGood]*s
        result = solve_subsistence_problem(store, FoodNutrients, my_p,␣
 ↪diet_min[group], diet_max[group], tol=tol)
        diet[my_p[ReferenceGood]] = result.diet

    Diet_response = pd.DataFrame(diet).T
    Diet_response.index.name = '%s Price' % ReferenceGood

    Diet_response.reset_index(inplace=True)

    # Get rid of units for index (cufflinks chokes)
    Diet_response['%s Price' % ReferenceGood] = Diet_response['%s Price' %␣
 ↪ReferenceGood].apply(lambda x: x.magnitude)

    Diet_response = Diet_response.set_index('%s Price' % ReferenceGood)

    # Just look at goods consumed in quantities greater than error tolerance
    return Diet_response.loc[:,(Diet_response>tol).sum()>0].iplot(xTitle='%s␣
 ↪Price' % ReferenceGood,
                                                                          ␣
 ↪yTitle='Quantity (Hectograms)',
                                                                          title=f"Change␣
 ↪in {good} Price vs Change in Diet Composition ({store}, {sex} {age})")
```

We chose "Whole Milk" as our reference good for the examples below, since we found it to be a recurring good across all generated recipes

[26]:
```
#example
price_quantity("Amazon Fresh", "M", "19-30", "Whole Milk")
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/dtypes/cast.py:1990:
UnitStrippedWarning:

The unit of the quantity is stripped when downcasting to ndarray.
```

1.10.3 3. Effect of Food Prices on Diet Nutrition

The `price_nutrition` function create a graph demonstrating **the relationship between the changes in food prices and the nutritional composition of the diet** for the specified store and sex-age group

[28]:
```
def price_nutrition(store, sex, age, good):
    group = f'{sex } {age}'
    FoodNutrients = match_n(store)
    Prices = match_p(store)
    tol=1e-6
    ReferenceGood = good

    scale = [0.5,0.75,0.9,1.,1.1,1.2,1.3,1.4,1.5,2,4]
    cost0 = solve_subsistence_problem(store, FoodNutrients, Prices,␣
↪diet_min[group], diet_max[group], tol=1e-6).fun

    my_p = Prices.copy()

    diet = {}
    for s in scale:
        my_p[ReferenceGood] = Prices[ReferenceGood]*s
        result = solve_subsistence_problem(store, FoodNutrients, my_p,␣
↪diet_min[group], diet_max[group], tol=tol)
        diet[my_p[ReferenceGood]] = result.diet

    NutrientResponse = pd.DataFrame(diet).T
    NutrientResponse.index.name = '%s Price' % ReferenceGood

    NutrientResponse.reset_index(inplace=True)

    # Get rid of units for index (cufflinks chokes)
    NutrientResponse['%s Price' % ReferenceGood] = NutrientResponse['%s Price'␣
↪% ReferenceGood].apply(lambda x: x.magnitude)
    NutrientResponse = NutrientResponse.set_index('%s Price' % ReferenceGood)

    # Matrix product maps quantities of food into quantities of nutrients
    NutrientResponse = NutrientResponse@FoodNutrients.T

    # Drop columns of missing nutrients
    NutrientResponse = NutrientResponse.loc[:,NutrientResponse.count()>0]
    return NutrientResponse.iplot(xTitle='%s Price' % ReferenceGood,
```

```
                                    yTitle='Hectograms',
                                   title = f"Change in {good} Price vs Change in␣
↪Nutritional Composition ({store}, {sex} {age})")
```

[37]:
```
#example
price_nutrition("Safeway", "M", "19-30", "Whole Milk")
```

1.11 [C] Visualization of Comparisons

[123]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-notebook')

import seaborn as sns
```

1.11.1 1. Bar Graph Comparison of the Cost Variation across Stores

The `cost_bar` function takes in an age range and generates a bar graph showing the different dietary cost of different stores for both male and female population of the specified age group.

Input Parameter:

- **age**: a str ("4-8", "9-13", "14-18","19-30","31-50", or "51+")

[225]:
```
def cost_bar(age):
    labels = ["Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl",␣
↪"Sprout's"]
    tol=1e-6

    def helper(group):
        p = []
        for i in labels:
            FoodNutrients = match_n(i)
            Prices = match_p(i)
            result = solve_subsistence_problem(i,␣
↪FoodNutrients,Prices,diet_min[group],diet_max[group],tol=tol)
            p += [round(result.fun,2)]
        return p
    male_p = helper(f'M {age}')
    female_p = helper(f'F {age}')

    x = np.arange(len(labels))  # the label locations
    width = 0.35
    fig, ax = plt.subplots()
    group1 = ax.bar(x - width/2, male_p, width, label=f'M {age}')
    group2 = ax.bar(x + width/2, female_p, width, label=f'F {age}')
```

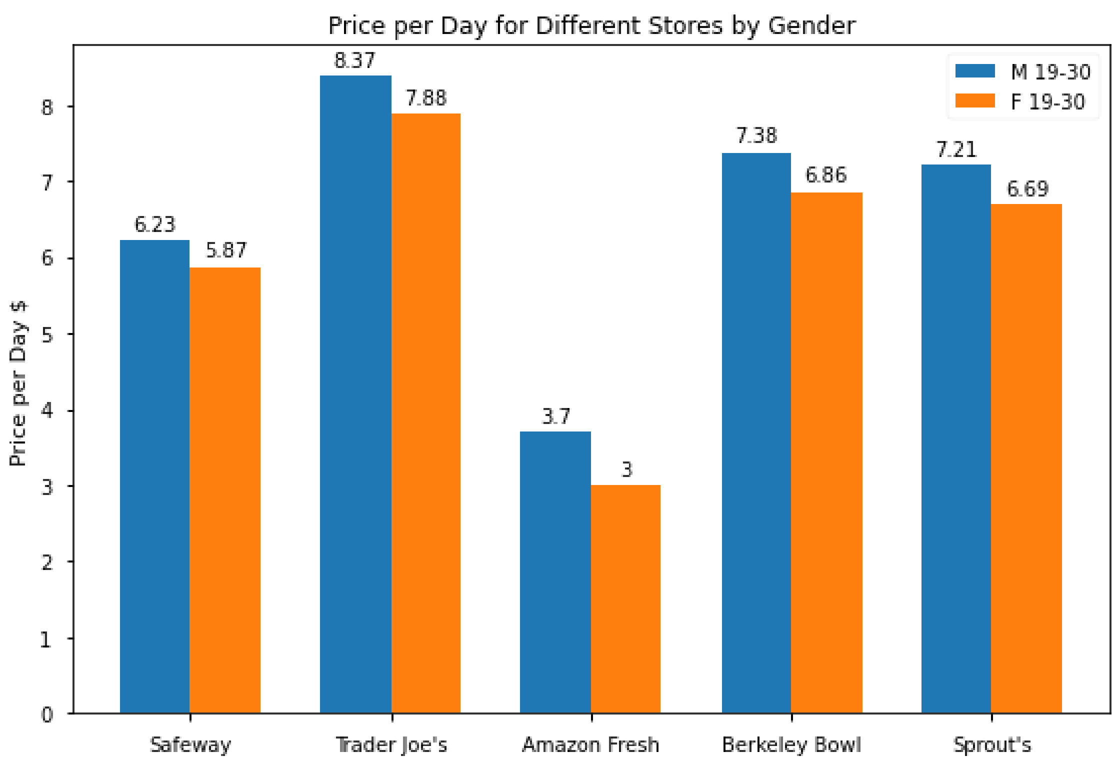
```
    ax.set_ylabel('Price per Day $')
    ax.set_title('Price per Day for Different Stores by Gender')
    plt.xticks(x, labels)
    ax.legend(frameon=True)

    ax.bar_label(group1, padding=3)
    ax.bar_label(group2, padding=3)
    fig.tight_layout()

    plt.show()
```

[226]:
```
#interactive plot to explore differences across age groups
# Observaton: the older the age group is, the larger the price difference␣
↪between Male and Female
interact(cost_bar, age = ["4-8", "9-13", "14-18","19-30","31-50", "51+"])
```

```
interactive(children=(Dropdown(description='age', options=('4-8', '9-13',␣
↪'14-18', '19-30', '31-50', '51+'), v…
```

[226]: <function __main__.cost_bar(age)>

[242]:
```
#example of a saved png
cost_bar("19-30")
plt.savefig('bar.png')
```



```
<Figure size 576x396 with 0 Axes>
```

1.11.2 2. Pie Chart Comparison of the Nutritional Contents across Stores

The `nutrition_pie` function takes the parameters and generates a pie graph showing percentages of different nutritions in descending order.

Input Parameter:

- **store**: a str with name of the store which is also the name of the individual sheet from the spreadsheet ("Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl", or "Sprout's")
- **sex**: a str ("M" or "F"
- **age**: a str ("4-8", "9-13", "14-18","19-30","31-50", or "51+")

[256]:
```python
def nutrition_pie(store, sex, age):
    group = f'{sex } {age}'
    FoodNutrients = match_n(store)
    Prices = match_p(store)
    tol=1e-6

    #cleaning data
    result = solve_subsistence_problem(store, FoodNutrients , Prices,␣
↪diet_min[group], diet_max[group], tol=1e-6)
    diet = result.diet
    test = np.abs(result.A).dot(diet)
    a = pd.DataFrame(data = test)
    a = a.sort_values(0, ascending=False)
    x = a.values.tolist()
    #flatten 2D list
    import itertools
    x = list(itertools.chain.from_iterable(x))
    #round amount to the nearest integer
    x = np.array([int(n) for n in x])
    my_lables = a.index.tolist()
    myexplode = [0.1, 0, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

    #creating pie chart
    colors = sns.color_palette("cubehelix_r", len(x))
    porcent = 100.*x/x.sum()
    patches, texts = plt.pie(x, colors= colors, startangle=90, shadow=True, ␣
↪radius=1.2, explode =myexplode)
    labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(my_lables, porcent)]


    sort_legend = True
    if sort_legend:
        patches, labels, dummy =  zip(*sorted(zip(patches, labels, x),
```

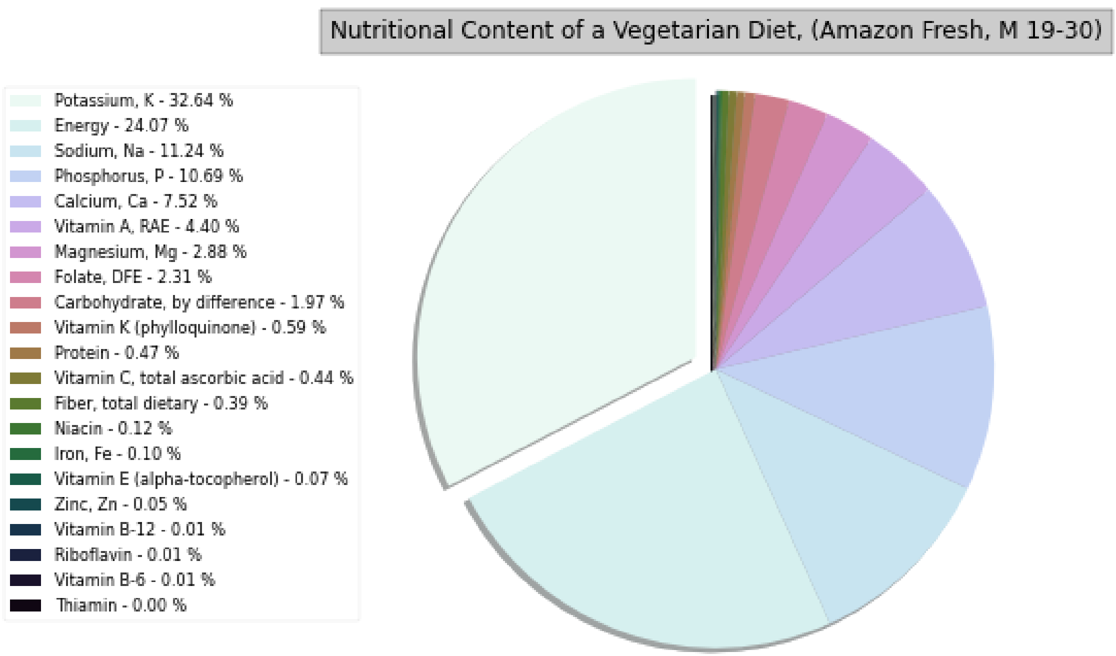
```
                                                   key=lambda x: x[2],
                                                   reverse=True))
    plt.legend(patches, labels, loc='best', bbox_to_anchor=(-0.1, 1.),
           fontsize=8)
    plt.title(f"Nutritional Content of a Vegetarian Diet, ({store}, {sex}␣
 ↪{age})", bbox={'facecolor':'0.8', 'pad':5}, y=1.05)
    plt.show()
```

[257]:
```
#interactive pie chart to explore differences across different stores, sex-age␣
 ↪groups
interact(nutrition_pie,
         store = ["Safeway", "Trader Joe's", "Amazon Fresh", "Berkeley Bowl",␣
 ↪"Sprout's"],
         sex = ["M","F"],
         age = ["4-8", "9-13", "14-18","19-30","31-50","51+"])
```

```
interactive(children=(Dropdown(description='store', options=('Safeway', "Trader␣
 ↪Joe's", 'Amazon Fresh', 'Berke…
```

[257]: `<function __main__.nutrition_pie(store, sex, age)>`

[265]:
```
#example of a saved png
nutrition_pie("Amazon Fresh", "M", "19-30")
plt.savefig('piechart.png')
```




```
<Figure size 576x396 with 0 Axes>
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```
## ignore the rest
```

[66]:
```
group = 'M 19-30'
solve_subsistence_problem("Safeway", safeway_n, diet_min[group],␣
↪diet_max[group], max_weight=None,tol=1e-6)
```

```
Key available for students@eep153.iam.gserviceaccount.com.
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/dtypes/cast.py:1990:
UnitStrippedWarning:

The unit of the quantity is stripped when downcasting to ndarray.
```

[66]:
```
      A:                              2% Milk  Almond Milk  Apples (fuji)
Arugula  \
Nutrition
Energy                           50.00        30.000          400.0  105.000
Protein                           3.33         0.380            0.0    2.580
Fiber, total dietary              0.00         0.200           20.0    1.600
Folate, DFE                       0.00         1.000            0.0   97.000
Calcium, Ca                     104.00       177.000            0.0  160.000
Carbohydrate, by difference       5.00         5.240           90.0    3.650
Iron, Fe                          0.00         0.270            0.0    1.460
Magnesium, Mg                     0.00         6.000            0.0   47.000
Niacin                            0.00         0.067            0.0    0.305
Phosphorus, P                     0.00         9.000            0.0   52.000
Potassium, K                    117.00        64.000            0.0  369.000
Riboflavin                        0.00         0.010            0.0    0.086
Thiamin                           0.00         0.000            0.0    0.044
Vitamin A, RAE                    0.00        86.000            0.0  119.000
Vitamin B-12                      0.00         0.000            0.0    0.000
Vitamin B-6                       0.00         0.000            0.0    0.073
Vitamin C, total ascorbic acid    0.00         0.000           12.0   15.000
Vitamin E (alpha-tocopherol)      0.00         2.700            0.0    0.430
Vitamin K (phylloquinone)         0.00         0.000            0.0  108.600
```

Zinc, Zn	0.00	0.060	0.0	0.470
Sodium, Na	-52.00	-69.000	-0.0	-27.000

	Avocado	Bananas	Bell Peppers	Beyond Burger	\
Nutrition					
Energy	160.000	312.00	17.00	1094.000	
Protein	2.000	12.50	0.68	14.850	
Fiber, total dietary	6.700	6.20	1.40	1.000	
Folate, DFE	81.000	0.00	0.00	89.000	
Calcium, Ca	12.000	125.00	14.00	76.000	
Carbohydrate, by difference	8.530	40.62	4.05	26.760	
Iron, Fe	0.550	1.12	0.49	2.870	
Magnesium, Mg	29.000	0.00	0.00	25.000	
Niacin	1.738	0.00	0.00	3.947	
Phosphorus, P	52.000	0.00	0.00	126.000	
Potassium, K	485.000	0.00	149.00	217.000	
Riboflavin	0.130	0.00	0.00	0.227	
Thiamin	0.067	0.00	0.00	0.334	
Vitamin A, RAE	7.000	0.00	0.00	0.000	
Vitamin B-12	0.000	0.00	0.00	0.000	
Vitamin B-6	0.257	0.00	0.00	0.101	
Vitamin C, total ascorbic acid	10.000	15.00	77.00	0.200	
Vitamin E (alpha-tocopherol)	2.070	0.00	0.00	0.040	
Vitamin K (phylloquinone)	21.000	0.00	0.00	5.500	
Zinc, Zn	0.640	0.00	0.00	2.380	
Sodium, Na	-7.000	-594.00	-27.00	-461.000	

	Black Beans	Broccoli	…	Tofu	\
Nutrition			…		
Energy	341.00	34.00	…	82.00	
Protein	22.73	2.70	…	8.24	
Fiber, total dietary	15.90	2.00	…	0.00	
Folate, DFE	0.00	0.00	…	0.00	
Calcium, Ca	136.00	41.00	…	118.00	
Carbohydrate, by difference	61.36	5.41	…	2.35	
Iron, Fe	4.09	0.73	…	1.69	
Magnesium, Mg	0.00	0.00	…	0.00	
Niacin	0.00	0.00	…	0.00	
Phosphorus, P	0.00	0.00	…	0.00	
Potassium, K	1477.00	0.00	…	188.00	
Riboflavin	0.00	0.00	…	0.00	
Thiamin	0.00	0.00	…	0.00	
Vitamin A, RAE	0.00	0.00	…	0.00	
Vitamin B-12	0.00	0.00	…	0.00	
Vitamin B-6	0.00	0.00	…	0.00	
Vitamin C, total ascorbic acid	0.00	89.20	…	0.00	
Vitamin E (alpha-tocopherol)	0.00	0.00	…	0.00	

Vitamin K (phylloquinone)	0.00	0.00	…	0.00
Zinc, Zn	0.00	0.00	…	0.00
Sodium, Na	-0.00	-54.00	…	-12.00

	Tomatoes (roma)	Tortillas	White Bread	\
Nutrition				
Energy	0.000000	214.00	269.00	
Protein	0.695625	3.57	11.54	
Fiber, total dietary	0.970600	3.60	3.80	
Folate, DFE	0.000000	0.00	0.00	
Calcium, Ca	9.963000	143.00	77.00	
Carbohydrate, by difference	3.837475	39.29	53.85	
Iron, Fe	0.103100	1.29	5.54	
Magnesium, Mg	8.089000	0.00	0.00	
Niacin	0.533100	0.00	0.00	
Phosphorus, P	19.090000	0.00	0.00	
Potassium, K	192.800000	0.00	0.00	
Riboflavin	0.000000	0.00	0.00	
Thiamin	0.055750	0.00	0.00	
Vitamin A, RAE	23.900000	0.00	0.00	
Vitamin B-12	0.000000	0.00	0.00	
Vitamin B-6	0.078940	0.00	0.00	
Vitamin C, total ascorbic acid	17.750000	0.00	0.00	
Vitamin E (alpha-tocopherol)	0.000000	0.00	0.00	
Vitamin K (phylloquinone)	0.000000	0.00	0.00	
Zinc, Zn	0.082450	0.00	0.00	
Sodium, Na	-0.000000	-0.00	-500.00	

	White Rice	Whole Milk	Whole Wheat Bread	\
Nutrition				
Energy	160.0	60.000	233.000	
Protein	3.2	3.280	11.630	
Fiber, total dietary	1.6	0.000	7.000	
Folate, DFE	0.0	0.000	0.000	
Calcium, Ca	0.0	123.000	47.000	
Carbohydrate, by difference	32.0	4.670	44.190	
Iron, Fe	0.0	0.000	2.510	
Magnesium, Mg	0.0	12.000	0.000	
Niacin	0.0	0.105	2.791	
Phosphorus, P	0.0	101.000	0.000	
Potassium, K	0.0	150.000	0.000	
Riboflavin	0.0	0.138	0.079	
Thiamin	0.0	0.056	0.000	
Vitamin A, RAE	0.0	32.000	0.000	
Vitamin B-12	0.0	0.540	0.000	
Vitamin B-6	0.0	0.061	0.000	
Vitamin C, total ascorbic acid	0.0	0.000	0.000	

```
Vitamin E (alpha-tocopherol)            0.0        0.050                0.000
Vitamin K (phylloquinone)               0.0        0.300                0.000
Zinc, Zn                                0.0        0.410                0.000
Sodium, Na                           -120.0      -38.000             -395.000

                                Yogurt (greek plain)  Zucchini  spaghetti
Nutrition
Energy                                        467.00     21.00    339.000
Protein                                         3.33      1.05     12.500
Fiber, total dietary                            3.30      1.10      8.900
Folate, DFE                                     0.00      0.00      0.000
Calcium, Ca                                   133.00     21.00     18.000
Carbohydrate, by difference                    70.00      4.21     75.000
Iron, Fe                                        0.00      0.44      3.040
Magnesium, Mg                                   0.00      0.00      0.000
Niacin                                          0.00      0.00      5.536
Phosphorus, P                                   0.00      0.00      0.000
Potassium, K                                    0.00    222.00    214.000
Riboflavin                                      0.00      0.00      0.357
Thiamin                                         0.00      0.00      1.000
Vitamin A, RAE                                  0.00      0.00      0.000
Vitamin B-12                                    0.00      0.00      0.000
Vitamin B-6                                     0.00      0.00      0.000
Vitamin C, total ascorbic acid                  0.00     12.60      0.000
Vitamin E (alpha-tocopherol)                    0.00      0.00      0.000
Vitamin K (phylloquinone)                       0.00      0.00      0.000
Zinc, Zn                                        0.00      0.00      0.000
Sodium, Na                                    -33.00     -0.00    -45.000

[21 rows x 45 columns]
       b: Nutrition
Energy                        2400.0
Protein                         56.0
Fiber, total dietary            33.6
Folate, DFE                    400.0
Calcium, Ca                   1000.0
Carbohydrate, by difference    130.0
Iron, Fe                         8.0
Magnesium, Mg                  400.0
Niacin                          16.0
Phosphorus, P                  700.0
Potassium, K                  4700.0
Riboflavin                       1.3
Thiamin                          1.2
Vitamin A, RAE                 900.0
Vitamin B-12                     2.4
Vitamin B-6                      1.3
```

```
Vitamin C, total ascorbic acid      90.0
Vitamin E (alpha-tocopherol)        15.0
Vitamin K (phylloquinone)          120.0
Zinc, Zn                            11.0
Sodium, Na                       -2300.0
Name: M 19-30, dtype: float64
     con: array([], dtype=float64)
    diet: 2% Milk                    5.136064e-11
Almond Milk             8.647422e-01
Apples (fuji)           4.216817e-11
Arugula                 1.361196e-11
Avocado                 2.754537e+00
Bananas                 1.712234e-09
Bell Peppers            3.047464e-12
Beyond Burger           2.211994e-12
Black Beans             7.358642e-01
Broccoli                1.453897e-11
Brown Rice              3.909143e-11
Butter                  3.720113e-12
Carrots                 9.264092e+00
Celery                  7.521266e-12
Cherries                3.381293e-12
Corn                    4.882657e-01
Cucumber                2.692659e-11
Eggs                    1.112693e-11
Grapes                  9.026091e-12
Hummus                  3.836136e-12
Ice Cream (vanilla)     7.418419e-12
Kale                    3.369712e-10
Lentils                 1.183615e-11
Lettuce (Romaine)       3.607363e-12
Mushrooms               3.526100e-12
Oats                    6.558002e-11
Onions (white)          1.207329e-11
Oranges                 6.336559e-11
Peaches                 6.366597e-12
Potatoes                4.283784e-01
Quinoa                  8.310254e-12
Rice Cakes              2.143705e-12
Spinach                 5.056852e-12
Strawberries            5.500833e-12
String Cheese           5.862649e-12
Tofu                    1.004485e-11
Tomatoes (roma)         1.094151e-11
Tortillas               1.220917e-11
White Bread             3.398826e-11
White Rice              5.097747e-11
```

```
Whole Milk               1.698007e+01
Whole Wheat Bread        2.990533e-11
Yogurt (greek plain)     1.148560e-11
Zucchini                 7.773344e-12
spaghetti                4.789865e-02
dtype: float64
     fun: 6.230496094593745
 message: 'Optimization terminated successfully.'
     nit: 26
   slack: array([1.50480446e-08, 3.23424543e+01, 2.36940736e+01, 1.76891035e-09,
       1.69561525e+03, 1.54337192e+02, 1.28746791e-10, 7.70228326e-10,
       4.33619363e-10, 1.49024913e+03, 3.49944545e+03, 1.96440437e+00,
       5.94766739e-01, 7.47252834e+03, 6.76923905e+00, 1.77997621e+00,
       1.65718461e-09, 1.29119826e-10, 6.52253129e+01, 2.31299424e-10,
       6.48306829e+02])
  status: 0
 success: True
       x: array([5.13606380e-11, 8.64742158e-01, 4.21681743e-11, 1.36119574e-11,
       2.75453727e+00, 1.71223389e-09, 3.04746390e-12, 2.21199440e-12,
       7.35864237e-01, 1.45389662e-11, 3.90914297e-11, 3.72011279e-12,
       9.26409154e+00, 7.52126618e-12, 3.38129256e-12, 4.88265693e-01,
       2.69265852e-11, 1.11269319e-11, 9.02609112e-12, 3.83613596e-12,
       7.41841876e-12, 3.36971175e-10, 1.18361504e-11, 3.60736286e-12,
       3.52610012e-12, 6.55800233e-11, 1.20732915e-11, 6.33655864e-11,
       6.36659671e-12, 4.28378415e-01, 8.31025400e-12, 2.14370461e-12,
       5.05685228e-12, 5.50083316e-12, 5.86264884e-12, 1.00448493e-11,
       1.09415144e-11, 1.22091709e-11, 3.39882605e-11, 5.09774677e-11,
       1.69800723e+01, 2.99053337e-11, 1.14855966e-11, 7.77334365e-12,
       4.78986500e-02])
```

[110]:

```python
#boron in mg/d
Boron = [3,6,6,11,11,17,17,20,20,20,20,20,20]
#Calcium in mg/d
Calcium = [2500,2500,2500,3000,3000,3000,3000,2500,2500,2500,2500,2000,2000]
#Copper in ug/d
Copper =␣
↪[1000,3000,3000,5000,5000,8000,8000,10000,10000,10000,10000,10000,10000]
Copper = [x/1000 for x in Copper]
#Iron mg/d
Iron = [40,40,40,40,40,45,45,45,45,45,45,45,45]
#Chloride (g/d)
Chloride = [2.3,2.9,2.9,3.4,3.4,3.6,3.6,3.6,3.6,3.6,3.6,3.6,3.6]
Chloride = [x*1000 for x in Chloride]
#vitamin A ug/d
vitA = [600,900,900,1700,1700,2800,2800,3000,3000,3000,3000,3000,3000]
vitA = [x/1000 for x in vitA]
# vitamin B6 mg/d
```

```python
vitb6 = [30,40,40,60,60,80,80,100,100,100,100,100,100]
#vitamin c mg/d
vitc = [400,650,650,1200,1200,1800,1800,2000,2000,2000,2000,2000,2000]
#viatmin d ug/d
vitd = [63,75,75,100,100,100,100,100,100,100,100,100,100]
vitd = [x/1000 for x in vitd]
#vitamin e mg/d
vite = [200,300,300,600,600,800,800,1000,1000,1000,1000,1000,1000]
#,Nutrition,Source,C 1-3,F 4-8,M 4-8,F 9-13,M 9-13,F 14-18,M 14-18,F 19-30,M 19-30,F 31-50,M 31-50,F 51+,M 51+
#0,"Sodium, Na",UL,1500,1900,1900,2200,2200,2300,2300,2300,2300,2300,2300,2300,2300
Directory = ['Sodium, Na','Boron','Calcium','Copper','Iron','Chloride','Vitamin A','Vitamin B6','Vitamin C','Vitamin D','Vitamin E']
bmax = pd.read_csv('./diet_maximums.csv').set_index('Nutrition').iloc[:,2:]
maxima_list = [Boron,Calcium,Copper,Iron,Chloride,vitA,vitb6,vitc,vitd,vite]

for nutrient in maxima_list:
    bmax.loc[len(bmax.index)] = nutrient
bmax['Nutrient'] = Directory
bmax = bmax.set_index('Nutrient')
bmax
```

[110]:

```
               C 1-3     F 4-8     M 4-8  F 9-13  M 9-13  F 14-18  M 14-18  \
Nutrient
Sodium, Na  1500.000  1900.000  1900.000  2200.0  2200.0   2300.0   2300.0
Boron          3.000     6.000     6.000    11.0    11.0     17.0     17.0
Calcium     2500.000  2500.000  2500.000  3000.0  3000.0   3000.0   3000.0
Copper         1.000     3.000     3.000     5.0     5.0      8.0      8.0
Iron          40.000    40.000    40.000    40.0    40.0     45.0     45.0
Chloride    2300.000  2900.000  2900.000  3400.0  3400.0   3600.0   3600.0
Vitamin A      0.600     0.900     0.900     1.7     1.7      2.8      2.8
Vitamin B6    30.000    40.000    40.000    60.0    60.0     80.0     80.0
Vitamin C    400.000   650.000   650.000  1200.0  1200.0   1800.0   1800.0
Vitamin D      0.063     0.075     0.075     0.1     0.1      0.1      0.1
Vitamin E    200.000   300.000   300.000   600.0   600.0    800.0    800.0

            F 19-30  M 19-30  F 31-50  M 31-50   F 51+   M 51+
Nutrient
Sodium, Na   2300.0   2300.0   2300.0   2300.0  2300.0  2300.0
Boron          20.0     20.0     20.0     20.0    20.0    20.0
Calcium      2500.0   2500.0   2500.0   2500.0  2000.0  2000.0
Copper         10.0     10.0     10.0     10.0    10.0    10.0
Iron           45.0     45.0     45.0     45.0    45.0    45.0
Chloride     3600.0   3600.0   3600.0   3600.0  3600.0  3600.0
Vitamin A       3.0      3.0      3.0      3.0     3.0     3.0
Vitamin B6    100.0    100.0    100.0    100.0   100.0   100.0
```

```
Vitamin C    2000.0   2000.0   2000.0   2000.0  2000.0  2000.0
Vitamin D       0.1      0.1      0.1      0.1     0.1     0.1
Vitamin E    1000.0   1000.0   1000.0   1000.0  1000.0  1000.0
```

[ ]:

```

```