The Stigler Diet problem is a classic optimization problem in linear programming, proposed by George Stigler in 1945. The goal of the problem is to find the least expensive combination of foods that meets a given set of nutritional requirements. Stigler used data on food prices and nutritional contents to identify a diet that would minimize the cost while satisfying the recommended daily allowances (RDAs) of various nutrients. However, it has the following limitations:

* Individual nutrient requirements can vary based on factors such as age, gender, activity level, and health status.
* It focuses solely on minimizing cost, disregarding factors such as taste, cultural preferences, and individual food aversions (like food allergies).
* It does not account for the importance of variety and a balanced diet.
* Food prices, availability, and nutritional content may change over time.
* It misses other essential nutrients, or non-nutrient factors that contribute to a healthy diet

To try to solve the Sigler diet problem, our group began datacrapping supermarket website using python libraries, getting the information, formatting the information, and use it to apply linear optimization:

Safeway: Dynamic website data scrapped with Selenium and Beautiful Soup:

* # #GETTING MEAT-SEAFOOD DATA
* # driver.get("https://www.safeway.com/shop/aisles/meat-seafood.3132.html")
* # # Wait for the dynamically loaded content to load
* **# time.sleep(5)** # This worked well for content, but other content had to be manually # loaded
* # # Use Selenium to scroll down the page and load more products
* # for i in range(3):
* # driver.execute\_script("window.scrollBy(0, 500);")
* # time.sleep(2)
* # # Get the HTML of the page and parse it with BeautifulSoup
* # # KEEP THESE DATA SAVED
* # html = driver.page\_source
* # meat\_seafood = BeautifulSoup(html, 'html.parser')
* # # Close the browser window
* # driver.quit()

As previously said, the dynamic website had to sometimes be loaded manually. A solution with Selenium was found:

# GETTING ALL MEAT AND SEAFOOD

* driver = webdriver.Chrome()
* driver.get("https://www.safeway.com/shop/search-results.html?q=meat")
* # Manually scroll down the page to the desired location
* **input("Scroll down to the desired location on the page, then press Enter to continue...")**
* # Get the HTML of the page and parse it with BeautifulSoup
* html = driver.page\_source
* all\_meat\_seafood = BeautifulSoup(html, 'html.parser')
* # Close the browser window
* driver.quit()

The code allowed for manually loading the website, which in turn allowed to collect all the food data.

The product’s name and respective price were found with:

* products\_names = all\_meat\_seafood.find\_all('a', {'data-qa': 'prd-itm-pttl'}, {'class': 'product-title\_\_name'})
* products\_prices = all\_meat\_seafood.find\_all('div', {'data-qa': 'prd-itm-pprc-qty'}, {'class': 'product-title\_\_name'})
* all\_meat\_seafood\_names = [name.text for name in products\_names]
* all\_meat\_seafood\_prices = [name.text for name in products\_prices]
* df\_all\_meat\_seafood = pd.DataFrame({'All Meat & Seafood Names': all\_meat\_seafood\_names, 'All Meat & Seafood Prices': all\_meat\_seafood\_prices})
* df\_all\_meat\_seafood

Some vegetable didn’t have prices, which cause some issues:

* products\_names = all\_veggies.find\_all('a', {'data-qa': 'prd-itm-pttl'}, {'class': 'product-title\_\_name'})
* products\_prices = all\_veggies.find\_all('div', {'data-qa': 'prd-itm-pprc-qty'}, {'class': 'product-title\_\_name'})
* all\_veggies\_names = [name.text for name in products\_names]
* all\_veggies\_prices = [name.text for name in products\_prices]
* # data cleaning via looking for missing price values (see below for the values found and cleaned # out (the names that didn’t have prices at the end were causing mismatch, NaN values were # introduced and discrepancies between food price and name were later eliminated):
* **# list1 = all\_veggies\_names**
* **# list2 = all\_veggies\_prices**
* **# if len(list1) < len(list2):**
* **# list1 += [float('nan')] \* (len(list2) - len(list1))**
* **# else:**
* **# list2 += [float('nan')] \* (len(list1) - len(list2))**
* **# df = pd.DataFrame({'col1': list1, 'col2': list2})**
* **# # Set display options to show all rows and columns**
* **# pd.set\_option('display.max\_rows', None)**
* **# pd.set\_option('display.max\_columns', None)**
* **# print(df)**
* # cleaning additional data for foods that have fuzzy/no price values
* del all\_veggies\_names[69]
* del all\_veggies\_names[157]
* del all\_veggies\_names[229]
* del all\_veggies\_names[293]
* del all\_veggies\_names[346]
* del all\_veggies\_names[363]
* del all\_veggies\_names[411]
* del all\_veggies\_names[431]
* df\_all\_veggies = pd.DataFrame({'All Veggies Names': all\_veggies\_names, 'All Veggies Prices': all\_veggies\_prices})
* df\_all\_veggies

The data was subsequently converted into several cvs files and concatenated into a big cvs file, where the price per mass was calculated:

# List of CSV files to concatenate

* csv\_files = ['all\_meat\_seafood.csv', 'all\_yogurt\_pudding.csv', 'all\_cheese.csv', 'all\_milk\_cream.csv', 'all\_butter\_sour\_cream.csv', 'all\_eggs.csv', 'all\_rice.csv', 'all\_grains.csv', 'all\_stuffing.csv', 'all\_pasta.csv', 'all\_fruits.csv', 'all\_veggies.csv']
* # List to store DataFrames for each CSV file
* dfs = []
* # Loop through CSV files and read them into a DataFrame, removing headers
* for csv\_file in csv\_files:
* df = pd.read\_csv(csv\_file, usecols=[0, 1], skiprows=[0], header=None)
* df.columns = ['Name', 'Price']
* dfs.append(df)
* # Concatenate DataFrames into a single DataFrame
* result = pd.concat(dfs)
* # Save concatenated DataFrame to a new CSV file
* result.to\_csv('concatenated\_file.csv', index=False)

How price per mass was calculated:

* # read in the csv file
* df = pd.read\_csv('concatenated\_file.csv')
* # function to extract float from string
* def extract\_float(text):
* try:
* return float(re.findall('\d+\.\d+|\d+', text)[0])
* except:
* return 0
* # create new column by multiplying values from columns 1 and 2
* df['Mass per money'] = df['Name'].apply(lambda x: extract\_float(x.split("-")[-1])).astype(float) \* df['Price'].apply(extract\_float).astype(float)
* # create fourth column based on 'Mass per money' column criteria
* df['Fourth Column'] = df['Mass per money']
* df.loc[df['Mass per money'] == 0, 'Fourth Column'] = df['Price'].apply(extract\_float).astype(float)
* # save updated dataframe to a new csv file
* df.to\_csv('UPDATED\_ALL\_DATA.csv', index=False)

Sadly, the safeway nutritional data has to be clicked to be obtained, and there are over 5k values!. There is an alternative solution. The US department of agriculture has a data dump file “2019-2020 FNDDS At A Glance - FNDDS Nutrient Values” in <https://www.usda.gov/> where the nutritional values of many foods are listed. It would be rather obvious that the names of foods will have to match with one another.

The solution is implementing he Levenshtein Distance. It is also known as the "Edit Distance”, which measures the similarity between two strings. It is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. It is used in natural language processing and computational biology to compare sequences and their similarities.

The goal, then will be to pair the most similar strings with the nutritional values:

* import pandas as pd
* import Levenshtein
* # Using the python-Levenshtein library to calculate the Levenshtein distance.
* # Read CSV files
* file\_F = pd.read\_csv('ALL\_FOOD\_PRICE.csv')
* file\_N = pd.read\_csv('stigler nutrional values.csv')
* # This code will read the CSV files, compare names using the Levenshtein distance,
* # Function to find the most similar name
* def find\_most\_similar\_name(name\_N, names\_F, matched\_names):
* best\_match = None
* best\_match\_index = None
* min\_distance = float('inf')
* for index, name\_F in enumerate(names\_F):
* if index not in matched\_names:
* distance = Levenshtein.distance(name\_N, name\_F)
* if distance < min\_distance:
  + min\_distance = distance
  + best\_match = name\_F
  + best\_match\_index = index
* return best\_match, best\_match\_index
* # Set to keep track of matched names
* # keep track of the names that have already been matched in file
* # ensure that a name is not matched more than once.
* matched\_names = set()
* # Iterate through each row in file\_N
* # If a name in file N has a closer match to a name that has already been matched in file F,
* # it will skip that name and find the next most similar name in file F that hasn't been matched yet.
* for index, row in file\_N.iterrows():
* name\_N = row['Main food description']
* most\_similar\_name\_F, most\_similar\_name\_index = find\_most\_similar\_name(name\_N, file\_F.iloc[:, 0], matched\_names)
* # Update matched\_names set
* matched\_names.add(most\_similar\_name\_F)
* # Append values from file\_N to the corresponding row in file\_F
* for col in file\_N.columns[1:]:
* file\_F.loc[file\_F.iloc[:, 0] == most\_similar\_name\_F, col] = row[col]
* # Save the updated file\_F to a new CSV file
* file\_F.to\_csv('FINALoutput.csv', index=False)

The mostly empty rows(the files that didn’t have any matches) are eliminated since we don’t gave nutritional values for them.

* # Keep rows with values in all specified columns
* specified\_columns = list(file\_N.columns[1:])
* filtered\_rows = file\_F.dropna(subset=specified\_columns)
* # Save the filtered rows to a new CSV file
* filtered\_rows.to\_csv('SECONDFINALoutput.csv', index=False)

Additionally, because the nutritional data is in terms of 100 g per unit food, it needs to be converted either into pounds or ounces:

* import pandas as pd
* import re
* def extract\_float(text):
* try:
* return float(re.findall('\d+\.\d+|\d+', text)[0])
* except:
* return 0
* # Read the CSV file
* df = pd.read\_csv('SECONDFINALoutput.csv')
* # Iterate through each row in the DataFrame
* for index, row in df.iterrows():
* name = row['Name']
* splitted\_name = name.split('-')
* if len(splitted\_name) > 1:
* multiplier = extract\_float(splitted\_name[-1])
* if 'L' in splitted\_name[-1].upper():
* multiplier \*= 0.220462
* elif 'O' in splitted\_name[-1].upper():
* multiplier \*= 3.5274
* # Multiply the values in the rest of the columns by the multiplier
* for col in df.columns[2:]:
* df.at[index, col] = df.at[index, col] \* multiplier
* # Save the updated DataFrame to a new CSV file
* df.to\_csv('THIRDFINALoutput.csv', index=False)

Now we can use linear optimization to find the stigler diet. Sadly, after many trials, we couldn’t come with a linear optimization alrogithm. Thankfully, the scipy library has one to finish our project.

* import numpy as np
* import pandas as pd
* from scipy.optimize import linprog
* # Read the CSV file
* df = pd.read\_csv('THIRDFINALoutput.csv')
* # Extract the objective function coefficients (, the second column named 'Fourth Column', has the #corresponding food prices)
* c = df['Fourth Column'].values
* # Extract the constraint matrix (nutritional contents: 'Energy (kcal)', 'Protein (g)',
* # 'Fatty acids, total monounsaturated and polyunsaturated (good fats) (g)',
* # 'Carbohydrate (g)', 'Fiber, total dietary (g)',
* # 'Vitamin A, RAE (mcg\_RAE)', 'Vitamin C (mg)', 'Calcium (mg)',
* # 'Iron\n(mg)', 'Vitamin D (D2 + D3) (mcg)',
* # 'Vitamin E (alpha-tocopherol) and added (mg)', 'Folate, total (mcg)',
* # 'Vitamin B-12, added\n(mcg)')
* A\_eq = df.iloc[:, 2:].values.T
* # Flip the signs of the inequalities to use less-than-or-equal-to constraints
* A\_ineq = -1 \* A\_eq
* # Constraint bounds (times 7 because we are looking at weekly compsuption, since going every #day to the store is time consuming, and buying in bulk may be cheaper)
* # I took the upper bound of the minimun requirements for an adult male who is very physically active
* b\_ineq = np.array([-3000\*7, -56\*7, -0.2 \* 3000\*7, -0.45 \* 3000\*7, -38\*7, -700\*7, -75\*7, -1000\*7, -8\*7, -600\*7, -15\*7, -400\*7, -2.4\*7])
* # Solve the problem
* res = linprog(c, A\_ub=A\_ineq, b\_ub=b\_ineq)
* # Print the results
* np.set\_printoptions(threshold=np.inf)
* # print("Optimal solution (Food quantities):", res.x)
* print("Minimum cost of the diet:", res.fun)
* # Print the food names, their corresponding non-zero quantities, and their prices
* food\_names = df['Name'].values
* food\_prices = df['Fourth Column'].values
* optimal\_quantities = res.x
* for name, price, quantity in zip(food\_names, food\_prices, optimal\_quantities):
* if quantity > 1e-6: # A small threshold to account for floating-point errors
* print(f"{name}: {quantity}, Price: {price}")