Report

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Spoonacular(https://spoonacular.com/food-api): collect price breakdown for each ingredient, store name, net price, and US unit
FoodData Central(https://fdc.nal.usda.gov/): collect Foundation food data, store every food's name, protein amount, and Vitamin amount

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

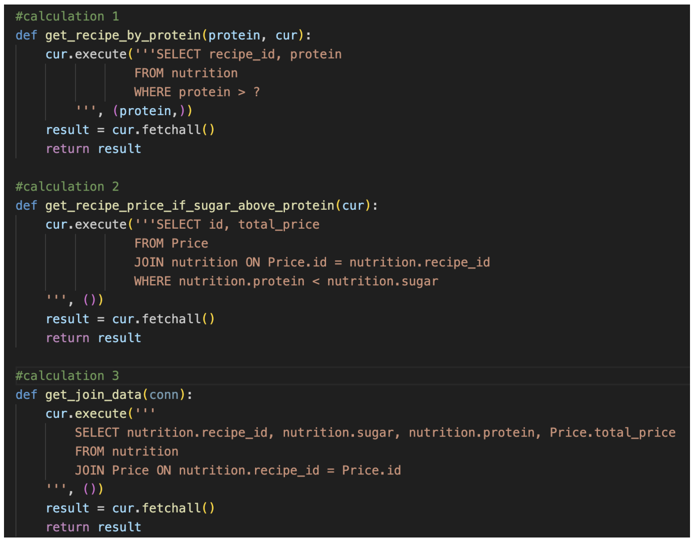
Spoonacular(https://spoonacular.com/food-api): collect price breakdown and nutritions for each recipe, store recipe id, total price, price per serving, protein amount, protein unit, sugar amount, sugar unit for each recipe

FoodData Central(https://fdc.nal.usda.gov/): collect Foundation food data, store every food's name, protein amount, and Vitamin amount

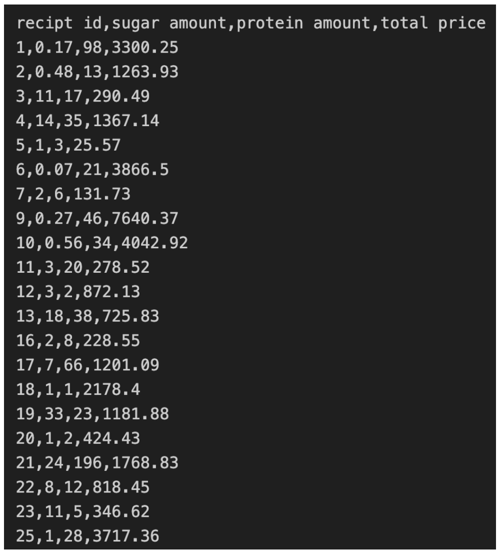
3. The problems that you faced (10 points)

The data in two APIs are not related so it is hard to get relationships between their data. Struggled with how to store at most 25 items each time.

4. The calculations from the data in the database (i.e. a screen shot) (10 points)



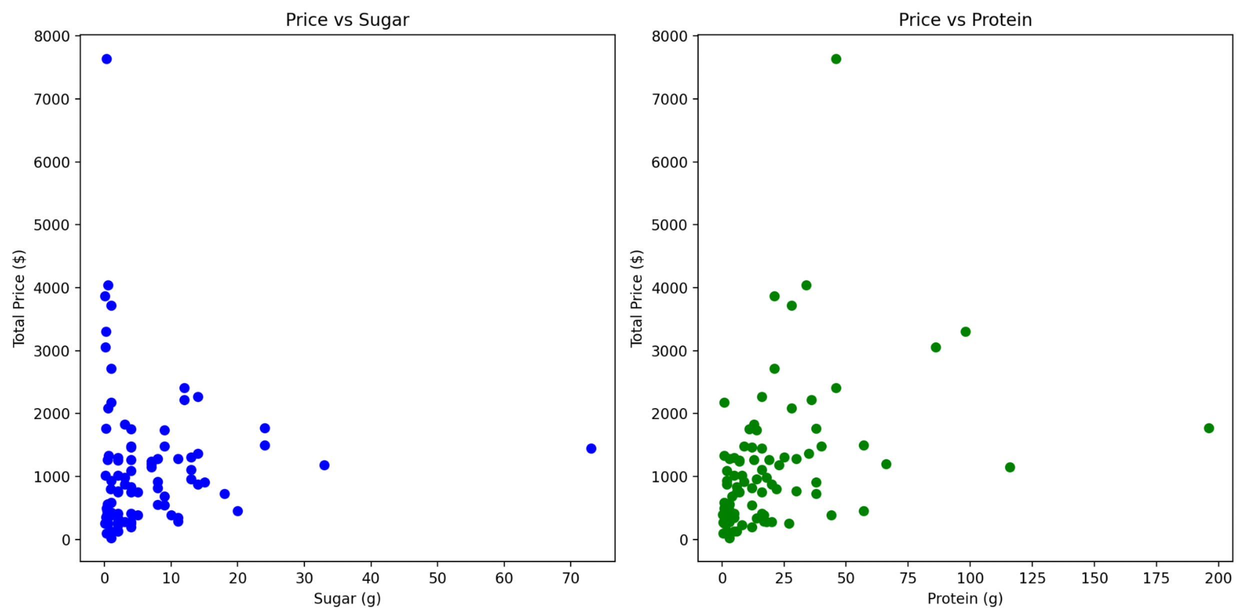
```
#calculation 1
def get_recipe_by_protein(protein, cur):
    cur.execute('''SELECT recipe_id, protein
                FROM nutrition
                WHERE protein > ?
        ''', (protein,))
    result = cur.fetchall()
    return result

#calculation 2
def get_recipe_price_if_sugar_above_protein(cur):
    cur.execute('''SELECT id, total_price
                FROM Price
                JOIN nutrition ON Price.id = nutrition.recipe_id
                WHERE nutrition.protein < nutrition.sugar
    ''', ())
    result = cur.fetchall()
    return result

#calculation 3
def get_join_data(conn):
    cur.execute('''
        SELECT nutrition.recipe_id, nutrition.sugar, nutrition.protein, Price.total_price
        FROM nutrition
        JOIN Price ON nutrition.recipe_id = Price.id
    ''', ())
    result = cur.fetchall()
    return result
```



```
recipt id,sugar amount,protein amount,total price
1,0.17,98,3300.25
2,0.48,13,1263.93
3,11,17,290.49
4,14,35,1367.14
5,1,3,25.57
6,0.07,21,3866.5
7,2,6,131.73
9,0.27,46,7640.37
10,0.56,34,4042.92
11,3,20,278.52
12,3,2,872.13
13,18,38,725.83
16,2,8,228.55
17,7,66,1201.09
18,1,1,2178.4
19,33,23,1181.88
20,1,2,424.43
21,24,196,1768.83
22,8,12,818.45
23,11,5,346.62
25,1,28,3717.36
```

Coding for three calculations what we got from calculation 3(saved as csv file)

5. The visualization that you created (i.e. screen shot or image file) (10 points)



6. Instructions for running your code (10 points)

1. Get data: Run foundation_food.py, nutrition.py, and price.py for 5 times, it will generate the final.db with Price, nutrition, proteins, insert_count and vitamins tables.
2. Process the data:
    a. run cal+join+vis.py to calculate how many receipts have protein amount above 10g, and how many receipts have sugar amount larger than protein amount, then save those results as a separate csv file called calculation.csv.
    b. At the same time, it also generates visualization.csv file with all data we need for visualization including recipe id, sugar amount, protein amount, and total price
    c. At the same time, it generates visualizations for the relationship between price and sugar amount and the relationship between price and protein amount.

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

Nutrition.py

write_json(filename, data)

Input:

filename: The name of the file where data will be written.

Data: The data to be written into the file.

Output:

Writes the provided data into the specified JSON file.

read_json(file_path)
Input:
file_path: Path to the JSON file to be read.
Output:
Returns a list of data items extracted from the JSON file.
create_table(cursor)
Input:
Cursor: Database cursor for executing SQL commands.
Output:
Creates a nutrition table in the SQLite database if it does not exist.
process_line(line, cursor)
Input:
line: A single record or line of nutrition data.
cursor: Database cursor to execute SQL commands.
Output:
Extracts sugar and protein values from the line and inserts them into the nutrition table in the database.
main()
Process:
Reads nutrition data from the JSON file.
Connects to a SQLite database (created if it doesn't exist).
Creates a nutrition table in the database.
Processes a batch of nutrition data and inserts it into the database.
Batch Processing:
It processes a batch of 25 records at a time, starting from the last processed ID, to avoid duplications.

Foundationfood.py
setup_database()
Output:
Sets up a SQLite database final.db with three tables: vitamins, proteins, and insert_count.
Returns a connection object to the SQLite database.
read_json_file(file_path)
Input:
file_path: Path to the JSON file to be read.
Output:
Returns parsed data from the JSON file.
get_insert_count(conn)
Input:
conn: Database connection object.

Output:

Retrieves and returns the current insert count from the insert_count table in the database.

update_insert_count(conn, new_count)

Input:

conn: Database connection object.

new_count: The new count value to be updated in the database.

Output:

Updates the insert count in the insert_count table in the database.

insert_food_get_id(cursor, food_name)

Input:

cursor: Database cursor to execute SQL commands.

food_name: The name of the food to insert.

Output:

Inserts a new food into the foods table and returns its ID.

insert_data_into_db(conn, data, limit=25)

Input:

conn: Database connection object.

data: Nutritional data to be inserted into the database.

limit: Optional. The number of items to process in one execution.

Output:

Inserts data into vitamins and proteins tables.

Updates the insert count in the database.

Main Workflow

main()

Process:

Sets up the database.

Reads nutritional data from a JSON file (foundationDownload.json).

Inserts the data into the database.

Closes the database connection.

Price.py

write_json(filename, dict)

Input: filename: str, the name of the file to write

Dict: dict, the dict data needed to be write into the file

Output: None

read_json(file_path)

Input: file_path: str, the name of the file to read

Output: dict: parsed JSON data from the file

set_up_database(db_name)

Input: db_name: str: the name of the database

Output: cur, conn: the database cursor and connection objects

set_up_price_table(data, cur, conn)

Input: data: list, data needs to be store in database

cur, conn: database cursor and connect

Output: None

main()

Get data from API and write them into a json file. Set database. Read the json file and write the data in the json file into the database table.

Cal+Join+Vis.py

get_recipe_by_protein(protein, cur)

Get data where the protein is above the given limit from the database

Input: protein: int, the protein amount limit

cur: database cursor

Output: list of tuples: [(recipe_id, protein), …]

get_recipe_price_if_sugar_above_protein(cur):

Get data where the sugar amount is larger than the protein amount from the database

Input: cur: database cursor

Output: list of tuples: [(recipe_id, total_price), …]

get_visualization_data(cur)

Input:

cur: Database cursor to execute SQL commands.

Output:

Executes a SQL query to join nutrition and Price tables and fetches data related to sugar, protein, and total price for each recipe. Returns the result as a list.

write_calculation_csv(x, y, filename)

Input:

x: First dataset to be written to the CSV.

y: Second dataset to be written to the CSV.

filename: The name of the CSV file where data will be written.

Output:

Writes the provided datasets x and y into the specified CSV file.

write_visualization_csv(z, filename)

Input:

z: Dataset to be written to the CSV.

filename: The name of the CSV file where data will be written.

Output:

Writes the dataset z into the specified CSV file.

Data Retrieval and CSV Writing

Database Connection:

Connects to the SQLite database final.db and creates a cursor for executing SQL commands.

Data Extraction:

Calls functions get_recipe_by_protein, get_recipe_price_if_sugar_above_protein, and get_visualization_data to retrieve various datasets from the database (the definitions for the first two functions are not provided in the script).

File Writing:

Writes the retrieved datasets into CSV files calculation.csv and visualization.csv for further analysis and visualization.

Data Visualization

Reading CSV:

Reads the data from visualization.csv using pandas.

Creating Scatterplots:

Generates two scatterplots to visualize the relationship between sugar and total price, and protein and total price.

Uses matplotlib for plotting, setting titles, labels, and layout.

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result
12/9/2023	Have difficulty writing the code for scatter plot using matplotlib	https://realpython.com/visualizing-python-plt-scatter/	The website gives the instruction for writing the code for scatterplot, and it solved the problem

Github repository link: https://github.com/SophieSu2723/final-project.git