Report for SI 206 Final Project

Goal

The goal of our project is threefold: obtaining a recipe with an inputted ingredient, calculate the cost associated with the recipe (using an API) and to analyze the cost of food in the past few years via inflation.

To be Achieved

What we achieved is very similar to our goal where we get a dish/food information from one of our APIs where the information includes its recipe and calculates the cost of the recipe. Thus, when a user types in one type of food such as chicken or beef the program returns possible recipes with that input. Afterwards, the program analyzes the cost, number of calories and what vitamins are in each recipe and then provides a small language translation of the recipe.

Problems Faced

We faced a few problems starting from when we actually started our project because Walmart closed its API during the time. As such we were unable to get enough information about many products. We then tried to use the API Kroger, but the authentication process was very complex. Also, we could not find an API that could help us calculation the inflation. Hence, we changed our plan slightly: We did not analyze the price of food in the past years, instead, we dove into nutrients of the food, while also providing a language translation feature.

One major problem we faced with language translation is that matplotlib graphs have a hard time recognizing many languages (possibly due to encoding problems), therefore, the places where we can put the translated text is very limited.

Calculations

Calculate.py: calculate the cost of recipes using join command.

Data stored in cost.txt

Visualization

Upon different user input, the program creates different types of graphs. Stored in “top_10_calories.png”, “top_10_cost.png”, “nutrients.png”. “nutrients.png” uses more complex visualization than others. Running the program each time will produce different graphs, as more data are inserted in the database.

Instructions

Run project_main.py. All programs are integrated in this main file.

a. Input a language such as English, Spanish, or Chinese. Invalid input will make the program default to English

b. Input what type of food you are interested in

c. Input the matrix you care about (will result in different graph, as well as different data inserted into the database)

Documentation

a. Calculate.py:

i. Calculate_cost_and_write()

Calculate the cost of recipes in the database relating to the specific product, write it to the file.

1. *Input:*
    a. language_abr – language of the program calculation output
    b. product – the product that the user is interested in
2. *Output*: Two files.

a. “cost.txt” contains the English version, which is subsequently used to graph the output.

b. “cost_translated.txt” contains the translated version

b. translation.py:

i. translateText()

Send request to the API and translate the given text based on the language

1. *Input:*
    a. language_abr – language of the program calculation output
    b. text – the text needs to be translated
2. *Output:* The translated text

ii. lang_abr()

Convert user input language to acceptable language abbreviation

1. *Input:* language – input language
2. *Output:* language abbreviation

c. graph.py:

i. graphTop10Calories():

Graph the recipes with the most calories in the database

1. *Input:* product – the product that the user is interested in
2. *Output:* Saved as “top_10_calories.png”

ii. graphTop10Cost():

Graph the recipes with the lease cost in the database

1. *Input:* costs - Dictionary of the costs of recipes
2. *Output:* Saved as “top_10_cost.png”

iii. graphNurtrients():

Graph random 20 recipes from the database with its nutrients information.

1. *Input:* product – the product that the user is interested in
2. *Output:* Saved as “nutrients.png”

d. Request_and_store.py:

i. Create_request_url():

Create urls needed for API request

1. *Input:* Product – the product that the user is interested in
2. *Output:* crafted URL for requesting API

ii. Request_recipe():

Request recipes from API

1. *Input:* food_search_url – url needed to request recipe from API
2. *Output:* A list of recipes

iii. SetupDatabase():

Set up, Stored data in the database

1. Input:
    a. Product- product the user is interested in

b. Recipe – A list of recipes of the product

2. Output: Set up the database at “Food Data.db”

iv. RequestCost():

Request API the cost of each ingredient of each recipe, store them in the database

1. Input:

   a. Product – product the user is interested in

   b. Ingredient-url – the URL needed to request the ingredient API

2. Output: Produce new table “Cost” in the new database, storing the ingredient costs of each recipe

Resources Used

Date	Issue Description	Location of Resource	Result
04/20	API for recipe search	https://developer.edamam.com/	Used
04/23	API for finding out cost of ingredients	https://spoonacular.com/food-api	Used
04/25	API for translation	https://cloud.google.com/translate/docs	Used
04/25	SQL Join + location of syntax	https://mode.com/sql-tutorial/sql-joins-where-vs-on/	Successfully Implemented
04/25	Matplotlib creates multiple bar graphs with color	https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html	Learned and used similar methods to implement
04/26	Integrates system and runs one python file with the other	https://www.tutorialspoint.com/How-can-I-make-one-Python-file-run-another	Learned and implemented

Important Notes:

1. The program limits the number of queries to each API to at most 20. For the ingredient-cost API, the program uses a counter in the loop to keep track of the number of queries, data already stored in the database will be directly retrieved and will not cost a query. For the recipes-search api, each search returns at most 20 recipes. On each run of the program, the starting index of each search is tracked via the max query id of the database.

2. The ingredient API https://developer.edamam.com/ limits Request 150 points/day, (approximately running the program 5 – 6 times/day). If it runs out, I have commented out several API keys("ingredient_key") associated with different gmail accounts I created for this API, so each one could be used to run another set of tests.

3. The google translation api https://cloud.google.com/translate/docs is essentially not free. We will have to limit run time used for the API.

4. Some ingredients cannot be found on the API. For those, a message will be printed, and the recipe with any ingredients not in the API will not be put in the "Cost" database, as the cost will not be accurate without any one of the ingredients information.

5. Each run of program will modify the database as well as the graph produced