SI 206 Final Project - Data Dynamos

Repository Link: https://github.com/shubhyadav10/SI-206-Final-Project.git

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

During our initial final project plan, we wanted to work with three shopping APIs: Facebook Marketplace, Amazon Marketplace Catalog, and eBay Analytics API to gather data related to the price, quantity, and reviews of certain items and calculate the relationship between the number of reviews a product has and the price of the product. Ultimately, upon further investigating these API's, we found that the approach we wanted to take would not entirely be possible and we pivoted to the food industry, focusing on fruit.

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

We ended up working with three API's: FruityVice, Recipe-Food-Nutrition, and Nutrition by API-Ninjas. We chose these three API's as they fit with our new goal of investigating the nutrition and cost content of fruits and the families they fall into. From the FruityVice API, we gathered the names, taxonomy, and basic nutritional information of the fruits in the API (with no standardized unit for nutritional information). From Recipe-Food-Nutrition, we gathered data on the cost of said fruits per 100g. Finally, from API-Ninjas, we gathered extended data on the nutritional data per 100g of said fruits.

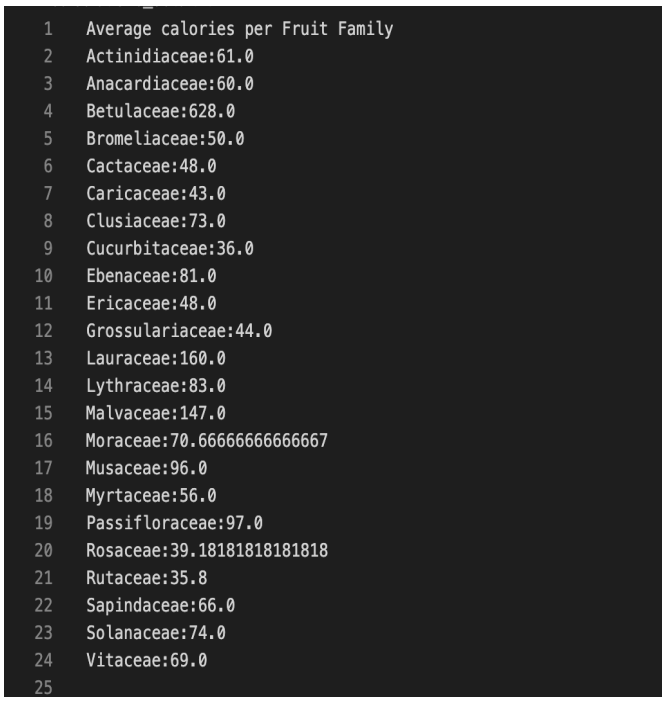
3. The problems that you faced (10 points)

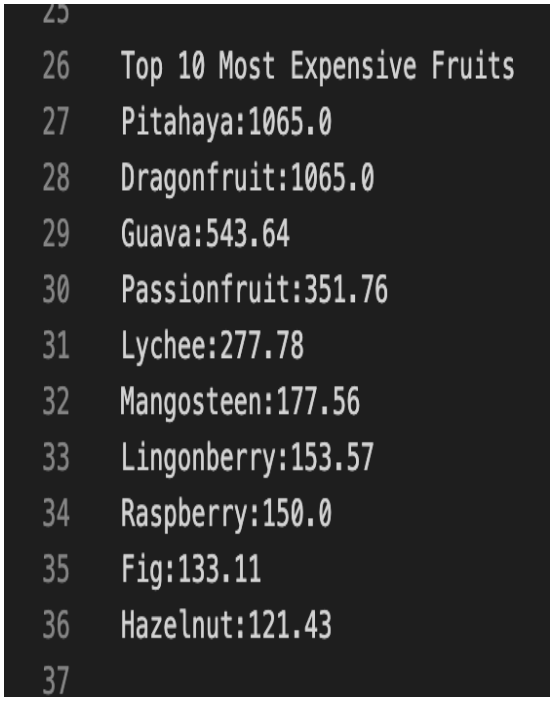
We encountered some problems along our way when working on our project. Our initial project plan of using Facebook Marketplace, Amazon Marketplace Catalog and eBay Analytics API was not feasible as we realized some of these API did not track dependent variables that we would have like to use in our project such as # of items sold for a said product. Thus, we pivoted to using the 3 Food API's which allowed us to track and compare fruit cost,calories and nutritional information extensively. We had wanted to compare inventory units for each fruit that we were tracking and comparing such that a customer could know what was available. However, we couldn't find any API that gave us this information for the fruits we currently had in our database. Therefore, we settled on using Nutrition by API-Ninjas to give an extended nutritional breakdown

of the fruits in our database. Our final problem came within our visualizations.py file wherein some of our plots seemed clustered with >30 values on the x-axis initially for a bar graph. To solve this problem, we ended up using “tight_layout” within matplotlib to give us better spacing and we inverted the bar graph using “barh” thus, making our plots more readable and understandable.

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

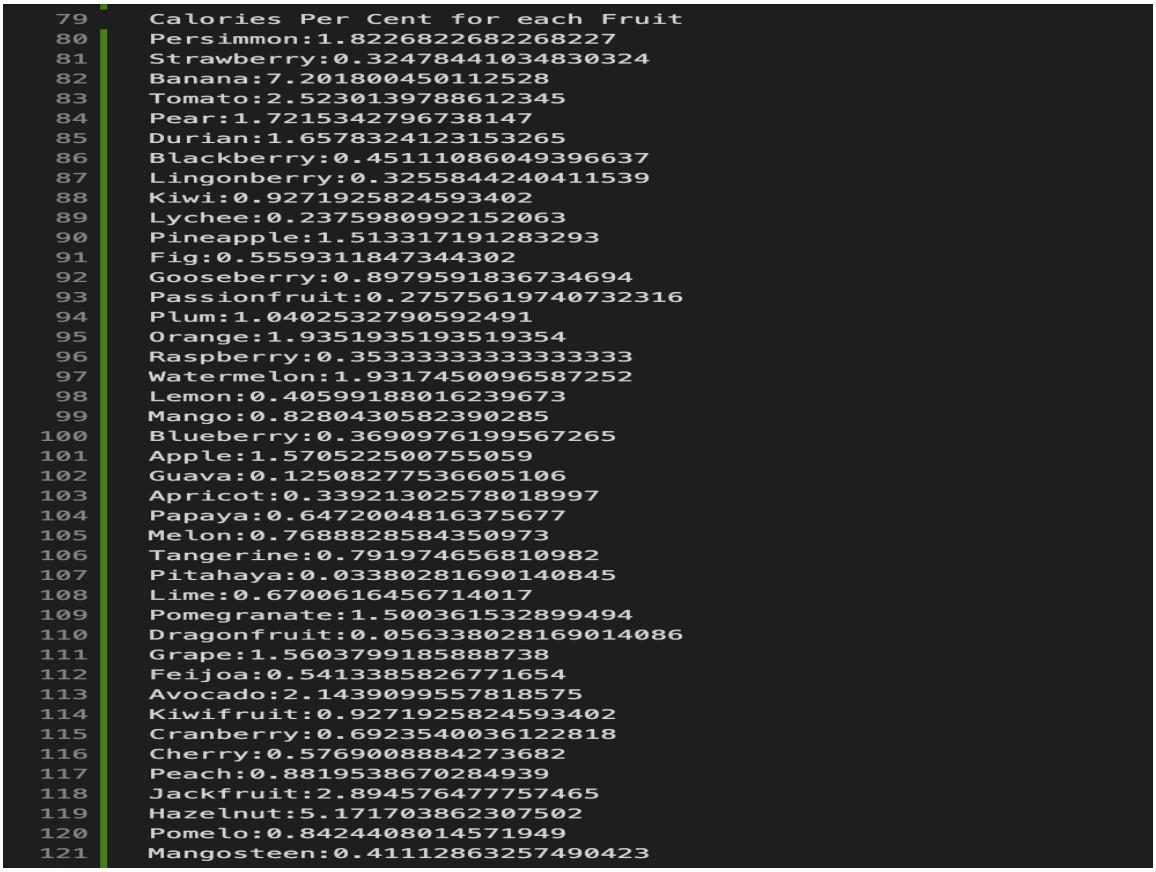
We had done 5 main calculations for our project, with each calculation eventually used in a visualization plot. The screenshots of the five calculations as indicated in the “calculations_data.txt” file are below:





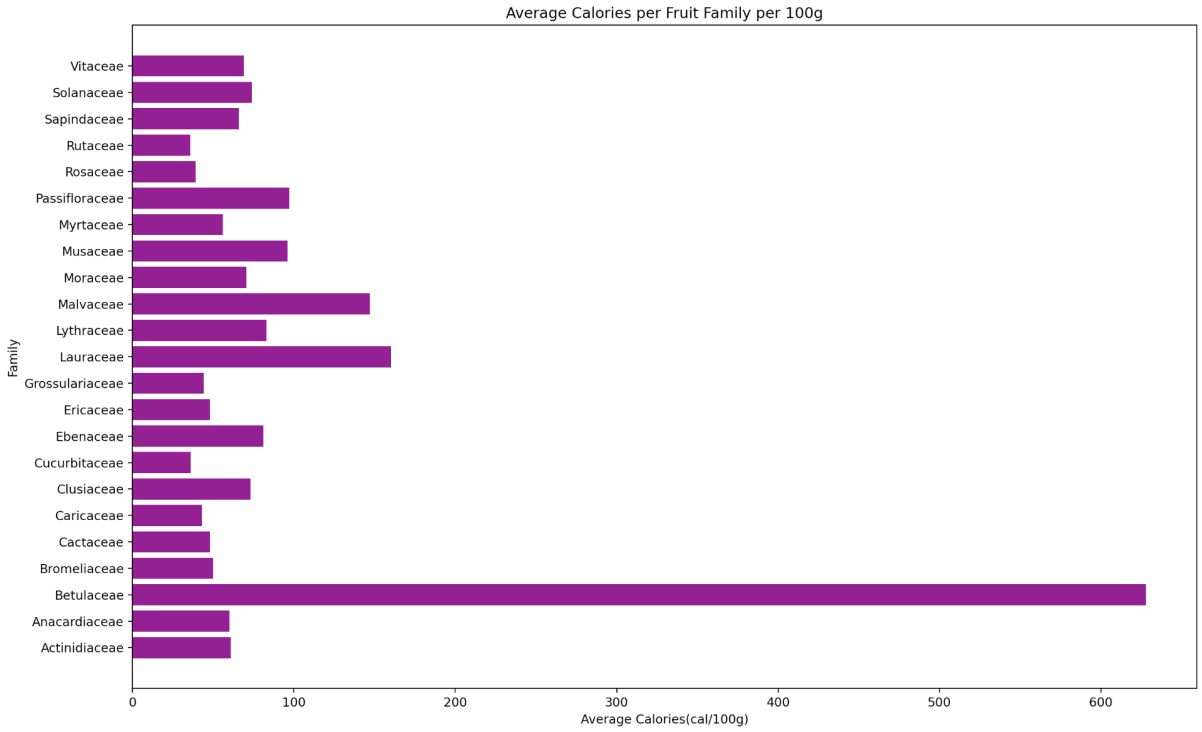
```
37
38      Top 10 Least Expensive Fruits
39      Banana:13.33
40      Watermelon:15.53
41      Orange:22.22
42      Tomato:29.33
43      Jackfruit:32.82
44      Pineapple:33.04
45      Pear:33.11
46      Apple:33.11
47      Lime:37.31
48      Pomelo:43.92
```

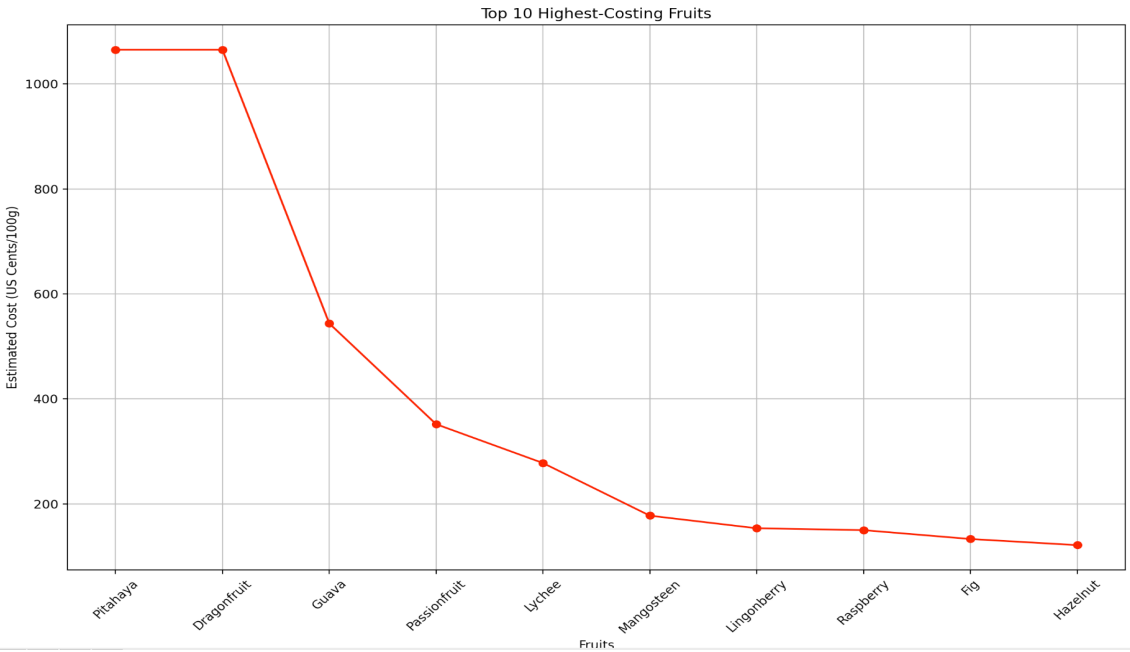
```
49
50  Nutritional content of each Fruit per 100g in terms of Carbs,Fat and Sugar content(g)
51  Banana:35.8
52  Orange:20.9
53  Strawberry:12.9
54  Pear:24.8
55  Tomato:6.699999999999999
56  Apple:24.6
57  Pineapple:23.0
58  Raspberry:16.9
59  Watermelon:13.7
60  Lemon:12.100000000000001
61  Mango:28.8
62  Blueberry:25.200000000000003
63  Apricot:20.8
64  Guava:24.200000000000003
65  Persimmon:31.6
66  Durian:32.7
67  Blackberry:15.0
68  Lingonberry:24.6
69  Kiwi:24.5
70  Lychee:32.5
71  Fig:36.099999999999994
72  Gooseberry:10.7
73  Plum:21.6
74  Cherry:28.900000000000002
75  Melon:16.3
76  Papaya:18.900000000000002
77  Lime:12.499999999999998
78
```

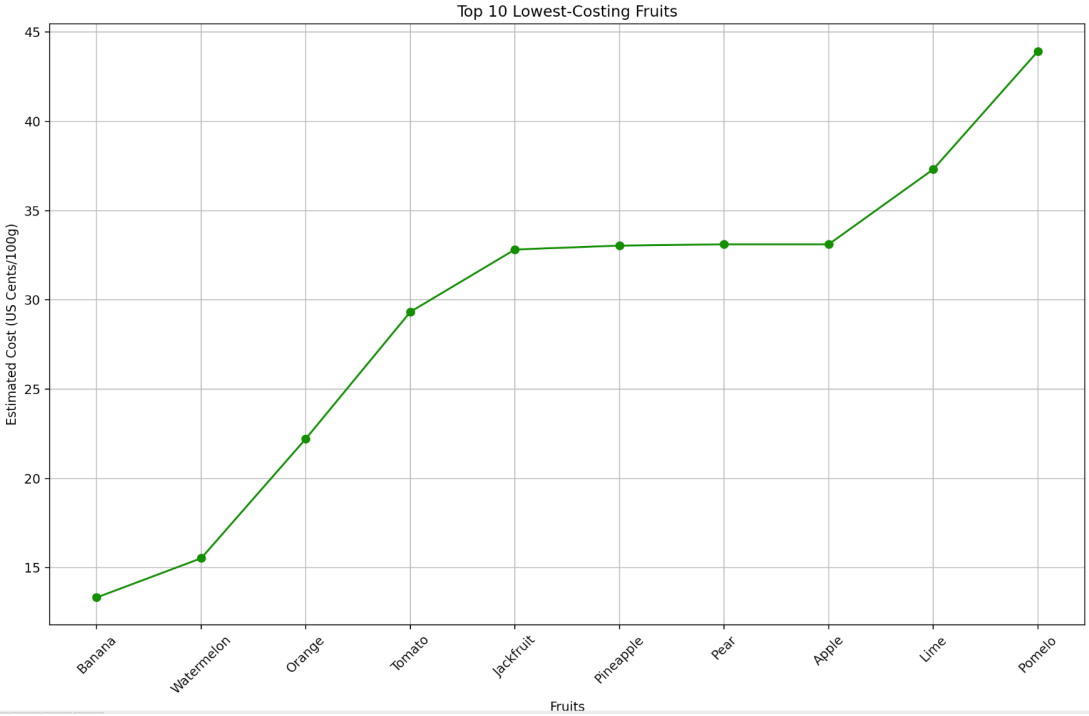


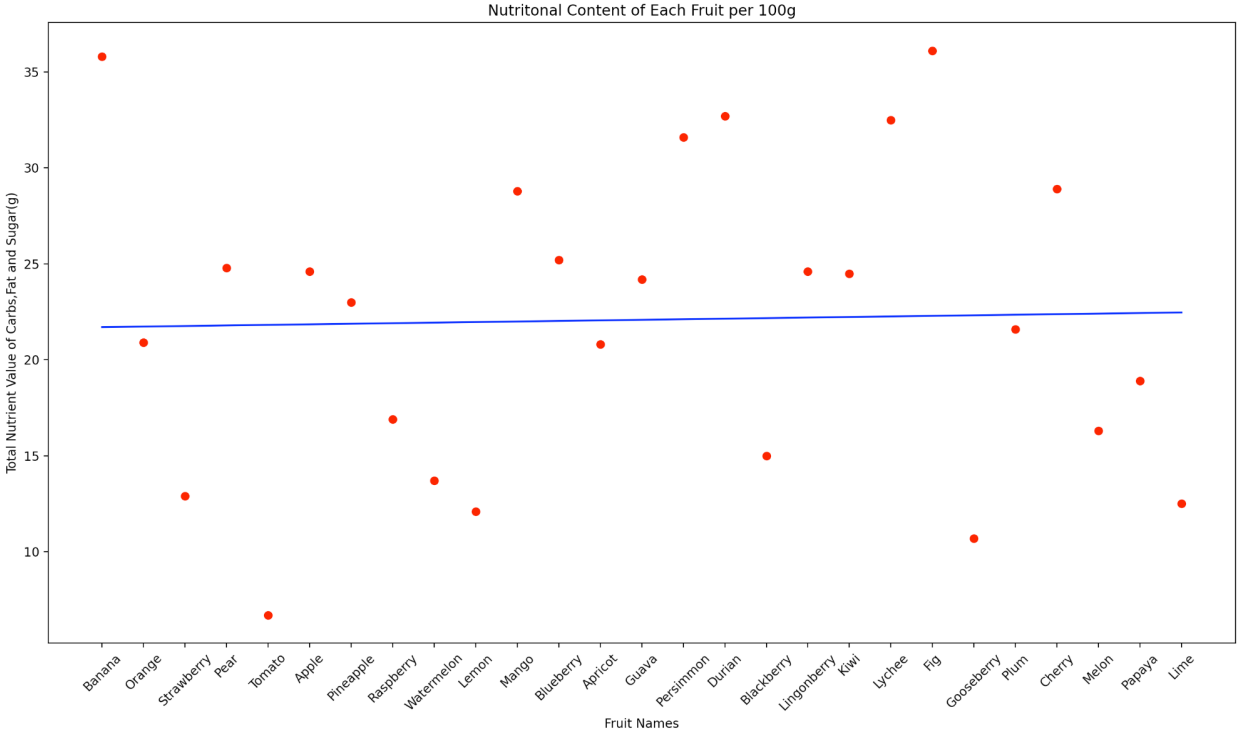
5. The visualization that you created (i.e screen shot or image file) (10 points)

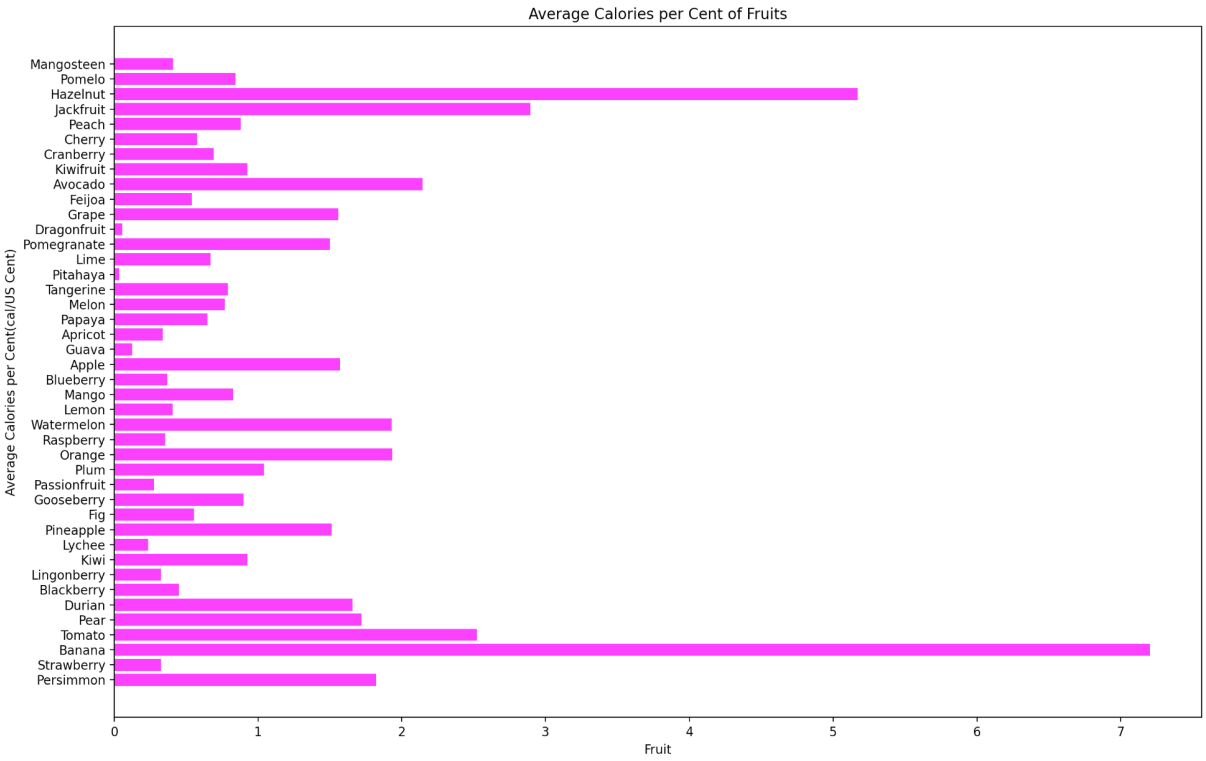
We had created 5 main visualizations for our project based on the data we had accumulated in our database. The corresponding calculations used for each visualization have been indicated above in question 4).


Average Calories per Fruit Family per 100g
Vitaceae
Solanaceae
Sapindaceae
Rutaceae
Rosaceae
Passifloraceae
Myrtaceae
Musaceae
Moraceae
Malvaceae
Lythraceae
Lauraceae
Grossulariaceae
Ericaceae
Ebenaceae
Cucurbitaceae
Clusiaceae
Caricaceae
Cactaceae
Bromeliaceae
Betulaceae
Anacardiaceae
Actinidiaceae
Family
0
100
200
300
400
500
600
Average Calories(cal/100g)


Top 10 Highest-Costing Fruits
1000
800
600
400
200
Estimated Cost (US Cents/100g)
Pitahaya
Dragonfruit
Guava
Passionfruit
Lychee
Mangosteen
Lingonberry
Raspberry
Fig
Hazelnut
Fruits


Top 10 Lowest-Costing Fruits
Estimated Cost (US Cents/100g)
45
40
35
30
25
20
15
Banana
Watermelon
Orange
Tomato
Jackfruit
Pineapple
Pear
Apple
Lime
Pomelo
Fruits


Nutritonal Content of Each Fruit per 100g
Total Nutrient Value of Carbs,Fat and Sugar(g)
35
30
25
20
15
10
Banana
Orange
Strawberry
Pear
Tomato
Apple
Pineapple
Raspberry
Watermelon
Lemon
Mango
Blueberry
Apricot
Guava
Persimmon
Durian
Blackberry
Lingonberry
Kiwi
Lychee
Fig
Gooseberry
Plum
Cherry
Melon
Papaya
Lime
Fruit Names



6. Instructions for running your code (10 points)

We have 3 files in our setup: code_1.py, extendednutrition.py and visualizations.py.

To get the output the setup has to be run in a particular order:

a) Run code_1.py for the first time
b) Run extendednutrition.py for the first time
c)Run code_1.py for the second time
d)Run extendednutrition.py for the second time
e) Finally, run visualizations.py to create all visualizations and also, the text file with the calculations

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

Code_1.py:

fetch_stored_fruit_ids(cursor): takes in a cursor object for fruityvice.db as input and returns a list of all fruit names in the Fruit table.

fetch_data(): has no arguments as input, just fetches all fruit info from the fruityvice api and returns a json object.

fetch_new_id_for_fruit(fruit_name): takes a fruit name as a string(taken from Fruit table), and returns its corresponding id number from the second api's database. Utilized api key to extract information from this api.

create_and_populate_new_table(fruit_names_with_new_ids): the input is a list of tuples of (fruit names,id's). All id's corresponding to the fruits here are those obtained from the second api. Creates a table NewFruitIDs with these two inputs as columns and an empty column estimated cost.

get_new_ids_from_db(): this function takes no input and returns a list of tuples of (fruitnames,id's) from the table NewFruitIDs created using the second api.

update_estimated_cost(fruit_data): takes in a list of tuples of (fruit names, id's). For every fruit, we use the corresponding id to calculate the cost in cents of 100 grams of this fruit by running requests on the api. We then update the table NewFruitIDs with the corresponding costs for each fruit.

Extendednutrition.py:

fetch_fruit_list(): takes no input, creates a cursor on fruityvice.db and selects and returns a list of all fruit names from the Fruit table.

extended_nutrition_data(fruit_name): takes a fruit_name string as input, and using an api key, gets corresponding nutritional info from the third api.

create_and_populate_extended_nutrition(fruit_names): takes a list of string fruit_names as input and does the following: creates a table ExtendedNutrition with the required columns for different nutrients, and for every fruit name, if information can be found on the third api's database, parses the returning result correctly and insert's a fruit with its nutrients as a row into the table.

Visualizations.py

avg_cal_fruit_family(): takes no input. It joins the Fruit and Nutrition table using the common integer id key. From here on, we select the fruit's family names by using GROUP and avg of calories within that fruit family, and put it into a dictionary with family

as key and average as the value. We plotted a horizontal bar graph with Fruit families on y axis and corresponding average calories on x axis.

highest_cost_graph(): takes no input. Selects the fruit names and costs of those 10 fruits with the highest costs in descending order from the NewFruitIDs table, and maps them into a dictionary with name as key and the costs as value. We plot these values on a linegraph, with fruit names on the x axis, and corresponding costs/100g on the y axis.

lowest_cost_grapth(): takes no input. Similar to the highest_cost_graph function, it selects the fruit names and costs of those 10 fruits with the lowest costs in ascending order from the NewFruitIDs table, and maps them into a dictionary with name as key and the costs as value. We plot these values on a linegraph, with fruit names on the x axis, and corresponding costs/100g on the y axis.

macro_fruit_scatter(): takes no input. Selects the fruit name, total fat(grams), carbohydrates(g), and sugar(g) from the ExtendedNutrition table. Adds the values of the fat, carbohydrates and sugars for a total nutritional value, and maps them into a dictionary with fruit name as the key and total nutritional value as the value. Creates a scatterplot using numpy in addition to matplotlib, charts fruit names on x axis and corresponding total nutritional value on the y axis.

plot_average_calories_per_cent(): takes no input. Joins tables from 2 api's by getting name from Fruit, calories from Nutrition, estimated cost per 100g from NewFruitIDs and joining fruit and Nutrition on the common id and fruit and NewFruitIDs on fruit name. Now, for whatever fruits from Fruit were available in NewFruitIDs, we calculated calories/estimated cost to get calories/cent, and mapped this into a dictionary with fruit name as value and calories/cent as value. Once again, we charted a horizontal bar graph with fruit names on y axis, and the corresponding calories/cent on x axis.

write_to_file_report(): takes no input. The function opens a file called calculations_data.txt, and writes the calculation values from all the 5 functions(which were stored in the dictionaries I mention about in each function).

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

Date	Issue	Location of Resource	Result (did it solve this issue?)

December 6th	Selecting our API's for our Food Project	https://github.com/public-apis/public-apis?tab=readme-ov-file#food--drink	Yes, the specific food and drink section amongst the public apis in Github allowed us to narrow down our APIs to our final three
December 7th	Status codes	https://support.oneskyapp.com/hc/en-us/articles/222410047-What-are-those-API-status-code-e-g-200-201-400-503-	Yes, we understood the different status codes an API can return on a query, and used them appropriately in our project.
December 7th	RapidAPI platform (Hosts Recipe-Food-Nutrition and Nutrition by API-Ninjas API's) and how to structure the queries	https://www.youtube.com/watch?v=AqTR8SHULGU&ab_channel=Rapid	Yes, the video details how the RapidAPI website can provide example queries based on parameters, as well as how to identify what parameters can be passed through, and what endpoints exist
December 8th	Querying data structure correctly from Nutrition by API-Ninjas	https://api-ninjas.com/api/nutrition	Yes, this documentation on the API gave me a better idea of what the data structure response would be and what parameters I would have to include in my query
December 8th	Costs were showing NULL in the NewFruitIDs table for estimated_cost	https://dba.stackexchange.com/questions/169458/mysql-how-to-create-column-if-not-exists	I understood then I would have to create the estimated cost column in

	column		NewfruitIDs at creation time, and then update costs as they were arsed from the second api instead of altering and adding the column and updating values, as running alter more than once gave all sorts of wrong stuff.
December 9th	Using line of best fit within a scatterplot for the 4th visualization	https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html	Yes, I learnt that I had to import numpy and use the polyfit() method for correct implementation of the line of best fit
December 9th	Decluttering the x-axis values for the 3rd and 5th visualizations	https://stackoverflow.com/questions/11264521/date-ticks-and-rotation	Yes, this forum introduced me to xticks which allowed me to customize the way I formatted x-axis values, increasing space by setting “rotation = 45”

Our github repository link(the canvas assignment shows to include a link to the repo in the report):

https://github.com/shubhyadav10/SI-206-Final-Project.git