Food Friends

Nina Busch and Tori Beckeman

https://github.com/vgbeck/final206.git

Goals

Original Goals:

Our original goal was to analyze two food related apis and look at data regarding number of ingredients per recipe, a pie chart with comparisons of the frequencies of types of cuisine, a bar chart comparing protein levels and sugar count in fruits, and a comparison across different types of fruit families for the average protein levels.

Achieved Goals:

After setting up the access to fruity vice and working with the data we discovered that it had less than 100 items. We chose to look at a different second API and analyze two different sets of data, one regarding meals and the other regarding holidays. Our achieved goals included Nina analyzing the average amount of ingredients in a set of 14 categories of meals as well as the percentage of meals in total that were vegetarian, vegan, and other to show how inclusive meals in this database were to those with common dietary restrictions. Tori analyzed the occurrences of holidays across a global calendar. This required data intake for 110 countries that included holding the name and country abbreviation for one table and the name, date, country abbreviation association etc for each holiday in the calendar. Because the overall holiday data for all countries would have over 1500 entries we chose to focus on four countries: US, France, Australia, Ireland and analyze the data of common holiday occurrences and what half of the year the majority of their holidays occur in.

Problems We Faced

1. The first problem we faced was with our apis. We started off with two food related apis, mealsdb (https://www.themealdb.com/) and fruity vice (https://www.fruityvice.com/). As stated in our goals, fruityvice did not hold enough data so we had to find a different database to use (https://date.nager.at/api). This is when we split into two questions that were interesting to us. Nina stuck with the meals database and Tori chose to analyze holiday data.

2. Another problem we faced was getting our different tables into one database. We had some github errors when pushing the database files and had to brainstorm the best way to

get all of the files in one place and access one database. To resolve this we only pushed the python files to the github and then used one person's computer to run the final code.

3. We struggled figuring out how to write to the same csv file. We had to make sure that we were not overwriting each other's data as it had to be pushed from two different places. Therefore, Tori used 'w' in her "with open()" statement and I had to use 'a' to write in append mode and not overwrite her data.

4. After figuring out problem #3, Nina ran into an issue where her data would append to the csv file multiple times if the code was run more than once.

   In order to fix this problem, I created a function that would check for the header in the csv file. If the header was there, I did not write to the file again, and vice versa.

5. Nina had trouble getting all of the information from the meals database.

   In order to get all of the information for each meal, I had to individually search up the categories and add each of the meals into a large json file called meals_by_id.json, which caused another problem mentioned next.

   I also had to search through all of the json files I made, which is why I made helper functions (all in helper_functions.py) to get the correct information into the database.

6. meals_by_id.json caused meals.py to take over a minute to run. Therefore, I added a statement to check if the path to the meals_by_id.json file didn't exist. If it did, I would not create the large json file over again, making the code take only about 6 seconds to run each time.

7. Tori had trouble getting her pie charts to appear in one visualization

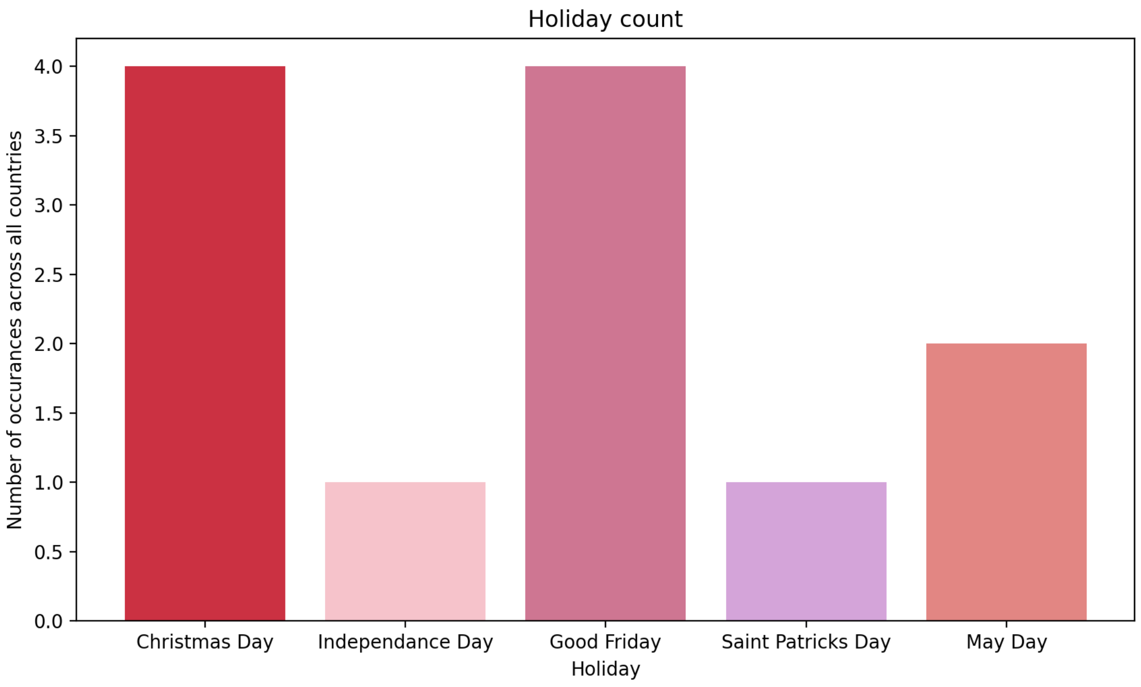
   To get the four country's pie charts to appear at once they had to be reorganized to be subplots. This required some research into different syntax and documentation of matplotlib to get a successful result.

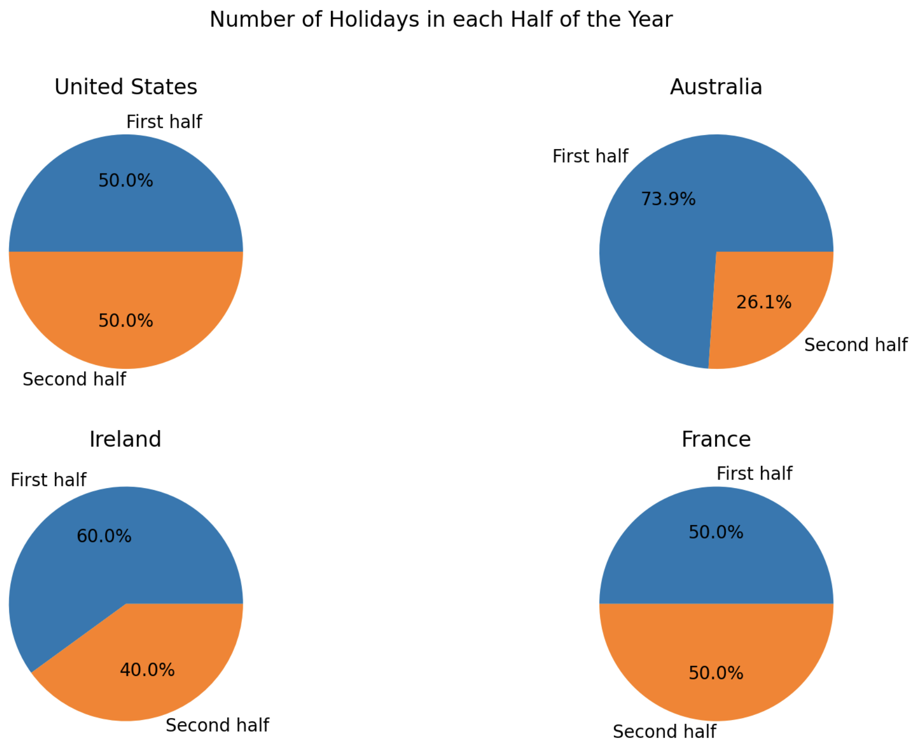
8. Tori did not have a good way to link the holiday and country tables without duplicate data

   The country's abbreviation (US, CA, etc) was the linking data that was present in both tables. For most of the project I worked with loading the data and the SQL files with each table holding the abbreviations. Upon realizing that this was duplicate data it had to get translated to using the unique identifier integer from the country table and saving that in the holiday table to keep track of the country by int instead of text. This required editing how the data was loaded into the database in the country.py file and editing the SQL calls in the holiday_visualize file

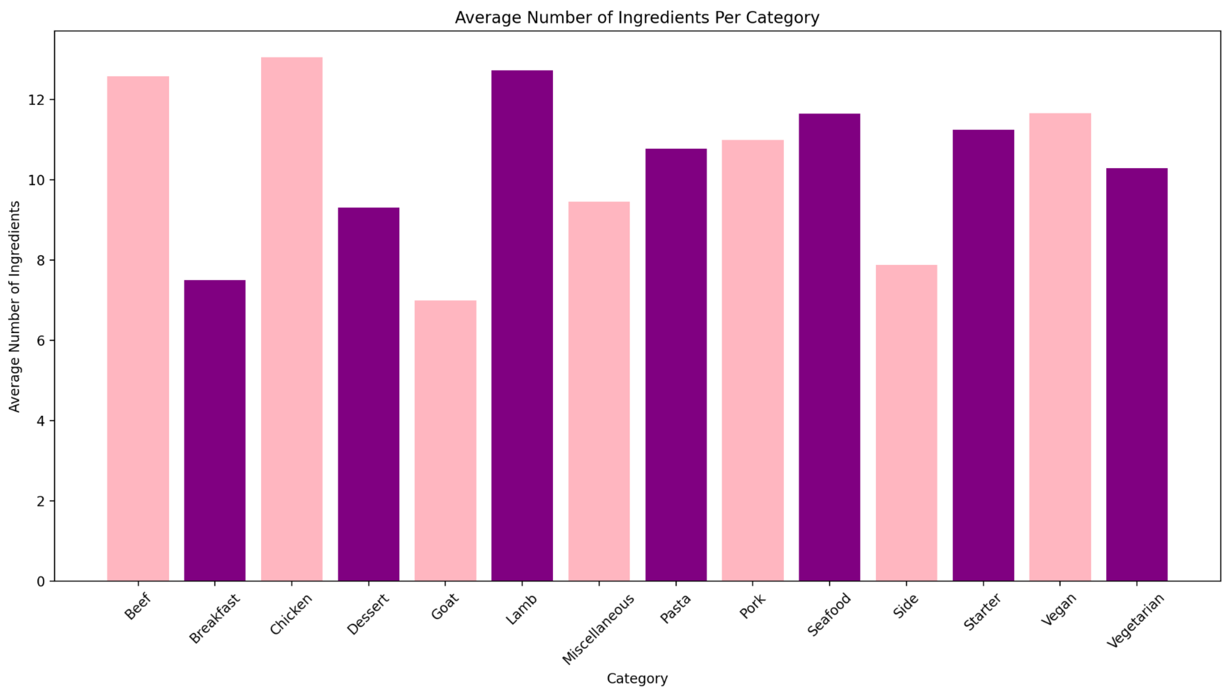
Visualizations

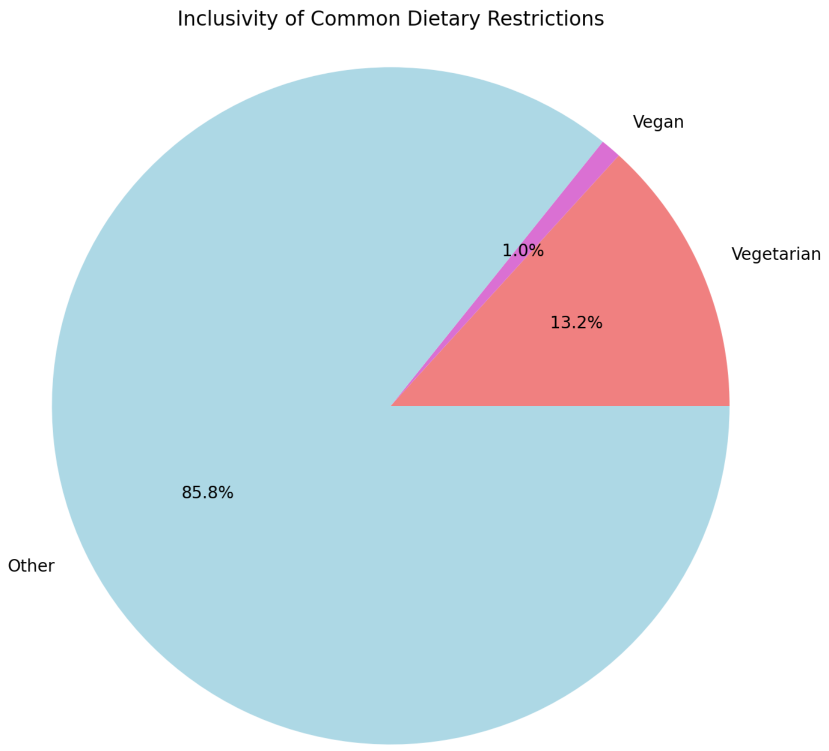
Tori - Holidays





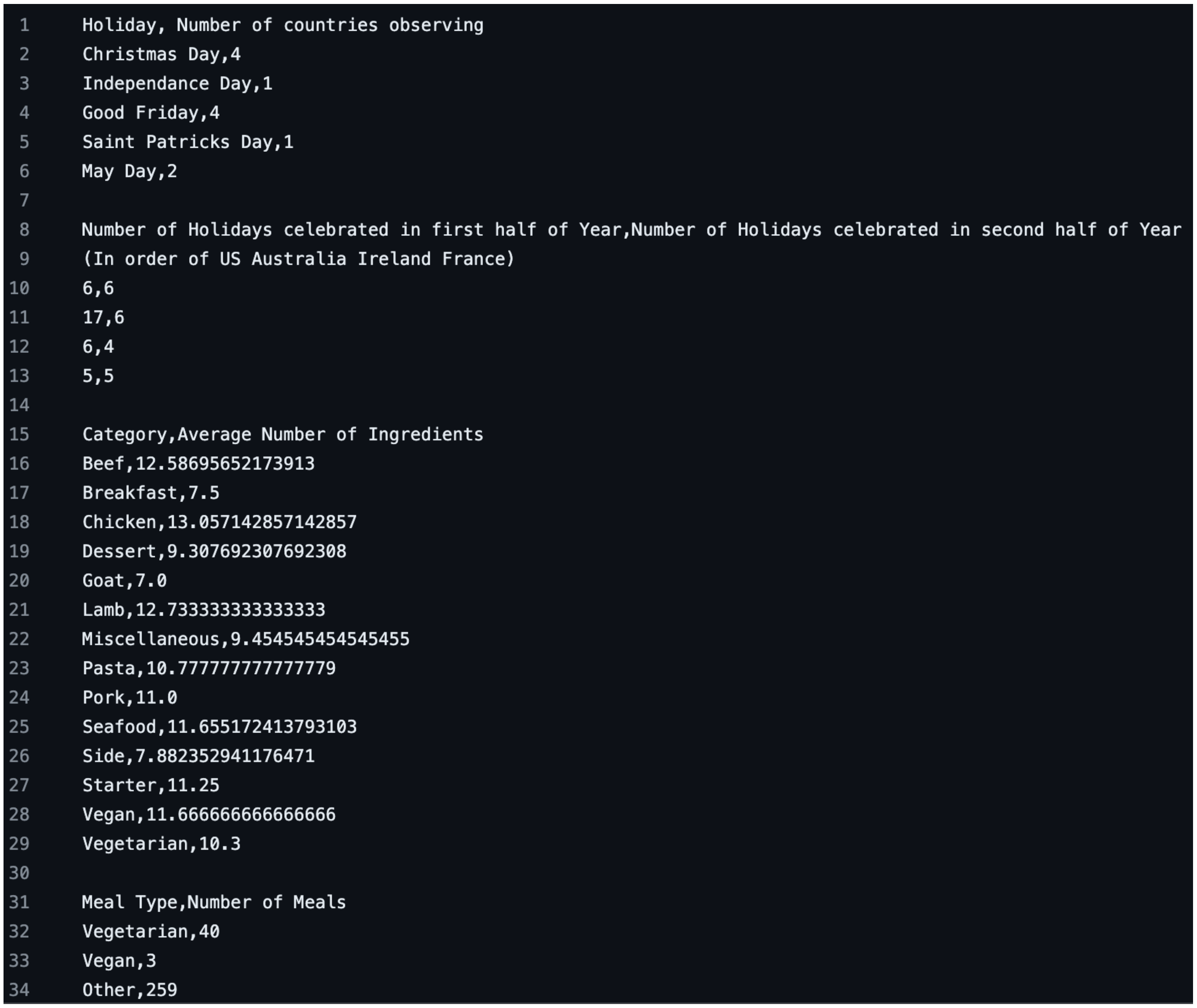
Nina - Meals


Average Number of Ingredients Per Category
Average Number of Ingredients
Category
0
2
4
6
8
10
12
Beef
Breakfast
Chicken
Dessert
Goat
Lamb
Miscellaneous
Pasta
Pork
Seafood
Side
Starter
Vegan
Vegetarian


Inclusivity of Common Dietary Restrictions
Vegan
1.0%
Vegetarian
13.2%
85.8%
Other

Calculations to CSV

Tori’s two visualizations followed by Nina’s two visualizations



```
Holiday, Number of countries observing
Christmas Day,4
Independance Day,1
Good Friday,4
Saint Patricks Day,1
May Day,2

Number of Holidays celebrated in first half of Year,Number of Holidays celebrated in second half of Year
(In order of US Australia Ireland France)
6,6
17,6
6,4
5,5

Category,Average Number of Ingredients
Beef,12.58695652173913
Breakfast,7.5
Chicken,13.057142857142857
Dessert,9.307692307692308
Goat,7.0
Lamb,12.733333333333333
Miscellaneous,9.454545454545455
Pasta,10.777777777777779
Pork,11.0
Seafood,11.655172413793103
Side,7.882352941176471
Starter,11.25
Vegan,11.666666666666666
Vegetarian,10.3

Meal Type,Number of Meals
Vegetarian,40
Vegan,3
Other,259
```

Instructions for running the code

There are 4 files to run. Follow these steps in order:

1. Run meals.py 13 times
2. Run country.py 5 times
3. Run holiday_visualize.py
4. Run meals_visualize.py

   *must run holiday_visualize before meals_visualize to get the correct csv output

Function Documentation

Tori's files

Country.py

- setUpDataBase(db_name)
  - Input: db_name, database name as a string
  - Process: Creates cur, conn, to access the database
  - Output: cur, conn
- create_country_table
  - Input: cur, conn
  - Process: Sets up the first table that holds a unique identifier, country name and abbreviation
  - Output: N/A
- add_country(filename, cur, conn)
  - Input: filename, cur, conn
  - Process: reads in the json of all countries. Adds the countries 25 at a time to the database. This is done by selecting the max id from the database and entering the next information 25 pieces of information from the country.json file
  - Output: N/A, modifies country table
- create_holiday_table(cur, conn)
  - Input: cur, conn
  - Process: sets up the second table that holds the holiday name, the unique identifier of the country it is associated with, the date, and local name

    - Output: N/A, creates holiday table
- add_holiday(cur, conn, holiday_count)
    - Input: cur, conn, holiday_count
    - Process: makes an execute call to find the max id of the holidays entered into the holiday table. Loops through 4 times and does the following once for each of the four chosen countries (United States, Australia, Ireland, France): checks to see if the country exists in the country table and if so checks that country's holidays are not already in the holiday table. If both are true it proceeds to make an API call to that specific country's holidays and stores the data into a json. This json is then loaded and looped through to save each holiday per country into the holiday table. There are never more than 25 holidays per country.
    - Output: N/A, adds data into the holiday table
- main()
    - Input: N/A
    - Process: calls to setupdatabase, create_country_tabe, writes the country data to a json, calls add_country, calls create_holiday_table, calls add_holiday.
    - Output: N/A

Holiday_visualize.py

- setUpDataBase(db_name)
    - Input: db_name, database name
    - Process: sets up cur, conn with database
    - Output: cur, conn
- christmas(cur, conn), independance_day(cur, conn), may_day(cur, conn),, friday_day(cur, conn), patrick_day(cur, conn)
    - Input: cur, conn
    - Process: each of these functions is a SQL call to find the list of countries that participate in the according holiday,
    - Output: a list of the names of the countries that celebrate (out of United States, Australia, Ireland, France)
- first_half_us(cur, conn), first _half_australia(cur, conn), first_half_ireland(cur, conn),first_half_france(cur, conn)
    - Input: cur, conn
    - Process: each of these functions is a SQL call to get the number of holidays in the according country that happen before 7/2
    - Output: A list of the holidays that happen in the country before 7/2
- second_half_us(cur, conn), second_half_australia(cur, conn), second_half_ireland(cur, conn), second_half_france(cur, conn)
    - Input: cur, conn
    - Process: each of these functions is a SQL call to get the number of holidays in the according country that happen after 7/2

    - Output: A list of the holidays that happen in the country after 7/2
- visualize(cur, conn)
    - Input: cur, conn
    - Process: This function calls the holiday functions and saves the len of what each one returns to make a bar chart representing the number of different countries that celebrate each of the 5 holidays
    - Output: A bar chart and returns a dictionary where the key is the holiday and the value is the number of countries that celebrates it out of France, US, Ireland, Australia
- visualize_two_pie(cur, conn)
    - Input: cur, conn
    - Process: this function calls the first_half/second_half functions and uses the length of them to make four pie charts representing the percentage of holidays in the first half of the year vs holidays in the second half of the year for four different countries
    - Output: Four pie charts in one display and returns a list of data per country that holds the number of holidays in the first half, and the number of holidays in the second
- main()
    - Input: N/A
    - Process: calls setup database, visualize, visualize_two_pie, and writes the output of the data from each of these to a csv file
    - Output: Writes the data used to create the bar and pie charts to output.csv

Nina’s files

helper_functions.py

- merge_category_files(files)
    - Input: files, a list of strings which are json file names
    - Process: This function reads a list of json files, each containing a list of dictionaries of meal data, and merges them into a single list, result, with all the data. Result is then written to a new file, ‘meals.json’ which contains all the meals, their names, ids, and links to photos.
    - Output: This function does not return anything, but it creates the file ‘meals.json’
- create_meals_by_id()
    - Input: None (but reads from ‘meals.json’ in the implementation)
    - Process: Reads data from ‘meals.json’, fetches additional meal details from the API that requires you to search each individual meal id, and accumulates this data in the list all_meals. The detailed data is then written to ‘meals_by_id.json’

    - Output: This function does not return anything, but it creates the file 'meals_by_id.json' with more detailed data than the json file created above, 'meals.json'
- get_num_ingredients(filename)
    - Input: filename, a string that is a json file (which is 'meals_by_id.json' when called) containing detailed meal data
    - Process: Reads the inputted json file and counts the number of non-empty ingredients for each meal
    - Output: Returns a dictionary where keys are meal_ids and values are the count of ingredients for each meal
- get_num_meals_per_category(filename)
    - Input: filename, a string that is a json file (which is 'meals.json' when called) containing less detailed meal data
    - Process: Reads the inputted json file and counts the number of meals in each category, appending these counts to a list, meals_count_list
    - Output: Returns the list meals_count_list where each element is the count of meals in a corresponding category
- get_category_id_dict(filename)
    - Input: filename, a string that is a json file (which is 'categories.json' when called) containing category data
    - Process: Reads the inputted json file and creates a dictionary mapping category names to their corresponding IDs
    - Output: Returns a dictionary where keys are the category names and values are their corresponding IDs
- get_category_id_list(filename)
    - Input: filename, a string that is a json file (which is 'meals_by_id.json' when called) containing detailed meal data
    - Process: Uses the above function, get_category_id_dict function to map category names to IDs for each meal in the inputted file
    - Output: Returns a list of category IDs in the order corresponding to meals in the inputted file
- count_data_points(filename)
    - Input: filename, a string that is a json file (which is 'meals.json' when called) containing all of the meals
    - Process: Reads the inputted json file and counts the total number of meals present in it
    - Output: Returns the total count of meals in the file, I used this to make sure that my API had more than 100 rows to be put into the database and did not use it elsewhere

meals.py

- set_up(db_name)
    - Input: db_name, the name of the SQLite database file to be created
    - Process: Sets up a connection to the SQLite database specified by db_name and initializes the cursor object for executing SQL commands
    - Output: Returns a tuple, (cur, con), where cur is the cursor object and conn is the database connection object
- create_meals_table(cur, conn)
    - Input:
        - cur – the SQLite cursor object used to execute SQL commands
        - conn – the SQLite connection object representing the database connection
    - Process: Creates a table named ‘meals’ in the SQLite database, with columns for ID, meal ID, name, category ID, and number of ingredients
    - Output: None, but creates the meals table
- create_categories_table(cur, conn)
    - Input: cur and conn, same as above
    - Process: Creates a table named categories in the SQLite database, with columns for category ID, category name, and the number of meals in that category
    - Output: None, but creates the categories table
- add_meal(filename, cur, conn)
    - Input: filename, which is a string that is a json file containing meal data (which is ‘meals_by_id.json’ when called), cur and conn are same as above
    - Process: Reads meal data from the inputted json file and inserts it into the ‘meals’ table in the database (or ignores it if it is already there). The data includes meal ID, name, category ID, and number of ingredients. The function inserts data into the database 25 at a time.
    - Output: None, but populates the ‘meals’ table in the database with the provided data
- add_category(filename, cur, conn)
    - Input: filename, which is a string that is a json file containing category data (which is ‘categories.json’ when called), cur and conn are same as above
    - Process: Reads category data from the inputted json file and inserts it into the categories table in the database (or ignores it if it is already there). The data includes category ID, name, and the number of meals in that category. It handles all 14 rows at once for each category
    - Output: None, but populates the ‘categories’ table in the database with the provided data

meals_visualize.py

- calculate_averages(cur)

    - Input: cur, same as above
    - Process: Executes a SQL command to calculate the average number of ingredients per category, joining the meals and categories table and groups the results by category name
    - Output: Returns a list of tuples, each containing a category name and the corresponding average number of ingredients
- create_barchart(avg_data)
    - Input: avg_data, a list of tuples which contains data of categories and their average number of ingredients, which is returned by the above function, calculate_averages
    - Process: Generates a bar chart visualization of the calculated average number of ingredients per category. The colors of the bars alternate between light pink and purple
    - Output: None, but displays the bar chart
- count_meal_types(cur)
    - Input: cur, same as above
    - Process: Counts the number of meals in specific dietary categories, Vegetarian and Vegan, as well as those that do not fall under these categories, titled 'Other'
    - Output: Returns a dictionary with the count of meals in each of the specified categories (vegetarian, vegan, and other)
- create_pie_chart(meal_counts)
    - Input: meal_counts, which is a dictionary returned by the above function count_meal_types, containing counts of meals by type (Vegetarian, Vegan, Other)
    - Process: Creates a pie chart visualization showing the distribution of meals across the specified dietary categories, using custom categories found from the stack overflow post linked below in our resource documentation
    - Output: None, but displays the pie chart
- file_contains_header(filename, header)
    - Input:
        - filename - the name of the csv file to check and
        - header - a list representing the header row to check for in the inputted csv file
    - Process: Opens the csv file and checks if the specified header row already exists in it
    - Output: Returns True if the header is found in the file, and False otherwise
- write_averages_to_csv(avg_data, filename)
    - Input:
        - avg_data - a list of types calculated from the above function calculate_averages
        - filename - the name of the csv file to write to

    - Process: Checks if the specific header is already in the file using the file_contains_header function, and if not appends the calculated average data for my first visualization to the csv file ‘output.csv’
    - Output: None, but appends data to a csv file
- write_num_meals_type_to_csv(meal_types, filename)
    - Input:
        - meal_types - a dictionary containing meal type counts calculator from the above function count_meal_types
        - filename - the name of the csv file to write to
    - Process: similar to write_averages_to_csv, checks if the specific header is already in the file and if not, appends the calculated meal type count data for my second visualization to the csv file ‘output.csv’
    - Output: None, but appends data to a csv file

Documentation of Resources

The resources used included the apis accessed, class materials (discussions, homeworks, lecture material), and ChatGpt to assist in coding syntax and appending text to a CSV file.

Date	**Issue Description**	**Location of Resource**	**Result (did it solve the issue?)**
12/7	Wanted to be creative in the color and design of our visuals	MatPlotLib colors	Yes, we chose colors based off of the names on this post
12/6	Wanted to write to the same csv files from multiple py files without overwriting	Appending CSV and ChatGPT	Yes, one of our files used ‘w’ the other used ‘a’
12/6	Wanted to make sure that the data wouldn’t append to the csv file every time the code was run	ChatGPT	Yes, we were able to add a function that checked if the header was in the file already
12/1	Outline for setting up a database, creating a table, and pushing items	Discussion 12	Yes, modeled our set up for the database on this example

12/8	Merge errors in Git	Git Merge	Yes, we were able to push and pull our data successfully
12/7	Wanted to plot 4 pie charts in one visual	Pie Charts	Yes, visualization was successful