College Cooks
Sam Feldman,
Marlee Hooberman,
Katie Radner

Final Project Reflection

1. What were your goals for the project?

MealsAPIFinal.py:

Our goals for the MealsDB API were:

To first collect meal inspiration ideas from the "filter by main ingredient" specification. We then planned to use this data to provide options to the user for what meal to make. We also planned to create a function that takes user input for an ingredient of choice, and have the function return names of meals using this ingredient.

We planned to calculate the count of meal names for each main ingredient provided. We also planned to calculate which main ingredients had the highest number of meal ideas, and which has the lowest.

Finally, we planned to create pie charts and/or scatter plots to compare the amount of meals with different ingredients from the total amount of meals we have from the MealsDB API.

100Diners.py:

Our goals for Top 100 restaurants were:

To collect data from the Top 100 Diners website. We wanted to collect all the data about the diner and organize it in the database. From here, our goal was to calculate either the average price for each type of restaurant or calculate the average number of reviews for each type of restaurant. We planned to display this data in a bar graph.

menu_nypl.py

Our goals for Menus were:

To become familiar with the database, once given access to the data with a token.

To extract all of the information from each dish found in menus that are stored in the API. Once this information was found, we planned on selecting a group of significant values pertaining to each dish that can be used for possible comparing/contrasting when in the database.

After the data was collected, the goal was to calculate the average prices of each dish, as the API gave the highest and lowest prices that the dishes were listed for on all the menus stored in the New York Public Library (NYPL), allowing for an opportunity to calculate an average price for a specific dish.

2. The goals that were achieved

MealsAPIFinal.py:

We were successfully able to collect different meals (which we intended to be meal inspiration ideas) from the "filter by main ingredient" specification from the MealsDB. We were able to organize this data in our database in the "Meals" datatable, in which the main ingredient used in the meals listed in the table is identified with a main ingredient id. This id was pulled from the complete list of ingredients on the MealsDB, utilizing the "List all Ingredients" specification. We identify each ingredient and its id in a separate table, the "Ingredients" table. We were then successfully able to calculate the counts of meal names for each main ingredient in the "Ingredients" table, using a query that joins the two tables together. We were able to then identify which ingredient had the highest number of meals, and which had the lowest (which may differ each time the code is run, based on which ingredients are randomly selected to

be the ones added to the meals table at runtime). Finally, we were successfully able to achieve our goal of creating a pie chart to compare the different amounts of meals for the different ingredients.

100Diners.py
We were successful in collecting the data for all 100 diners on the website. Using BeautifulSoup, the data was scraped off of the website and cleaned up in order to be put in the database. Once all the data was collected and cleaned, we organized it in the database by name of restaurant, type of restaurant, amount of money it generally costs, amount of customer reviews, and location by city. It is easily accessible to understand and follow. After everything was put in the table correctly, we created a visualization to poraty the average amount of money it costs to eat at a certain type of restaurant. These numbers were calculated off of "$" signs. Four dollar signs in a row was the most expensive, while one dollar sign was the least expensive. The bar graph depicts the average dollar signs for each type of restaurant, showing, on average, how much a consumer would spend at a certain type of restaurant.

menu_nypl.py
With success, we were able to reach out to the NYPL and receive access to the Menus API with a personal token. Once access was given, we were able to see the types of data we could scrape from the API, including the action we had hoped to be able to complete: finding the types of dishes found in the data collection. After producing code to access this particular sector of the API, we were able to find nested dictionaries and lists within an overall dictionary containing the dishes and specific features about each one. We achieved our goal of selecting and identifying several of the features, loading them into the database; the table consists of an id number of each dish, along with its name, the number of menus and times it has appeared, when it first and last appeared, and its highest and lowest prices as collected from all of the menus in the API. Continuing, the extracted data allowed for the calculation of the average price per dish, adding the two prices and dividing it by two. Once these calculations were made for every dish that had both prices included from the API, we achieved the goal of creating an interactive bar graph conveying the top ten highest average prices of the dishes.

3. The problems that we faced

MealsAPIFinal.py:
A problem we faced with the Meals DB was utilizing a function to have a user input which ingredient they would like to see the meals for. Although we were successfully able to create this function, it did not align with the goals of the assignment specifically to have 25 new meals entered each time the code is run. So, rather than ask the user for input for the ingredients, we have the ingredients randomly generated.

100Diners.py
One problem we encountered was with a different website deemed LesserEvil. We originally planned to scrape data from this website and portray the nutritional information and calculate its averages for each product. However, we later realized this website did not have 100 items to add into the database, therefore, it would not fit the project requirements. To resolve this issue, we switched the website to Top 100 Diners to make sure we had 100 items to add.

menu_nypl.py

A problem we faced was with a different API: FruityVice. With FruityVice, we had planned to extract the nutritional information found from a significant amount of fruits (over 100). However, the API only contained nutritional information for 23 fruits; this was not enough to fulfill the requirement of having at least 100 items in the database. Thus, we had to turn to a different API, leading us to find NYPL's Menus API. Once this API was found, we were able to successfully achieve our goals and fulfill the requirements for this API. Another problem faced with the Menus API was figuring out how to enter the specific value of 25 new dishes every time the code was run. After receiving guidance, this problem was solved. Lastly, not all of the dishes in the API had the highest/lowest prices listed, so we had to figure out a way to select the data that *only* had prices listed in order to successfully perform desired calculations on the numbers.

4. **Your file that contains the calculations from the data in the database (10 points)**

MealsAPIFinal.py:
The screenshot below and the above link show example output to this .txt file.

```
Calculations from The Meals DB API
Ingredient with most meals,Ingredient with least meals
Egg Yolks,Worcestershire Sauce

Top 10 Main Ingredients Based Off Of Counts Of Meals:
Ingredient,Count of Meals With That Main Ingredient
Egg Yolks,13
Sugar,7
Chickpeas,6
Peas,5
Brandy,5
Olive Oil,5
Kale,5
Greek Yogurt,5
Vegetable Stock,5
Soy Sauce,4
```

100Diners.py
The screenshot below shows the final output of the restaurants_calculations.txt file:

restaurant_calculations.txt

```
1  The average cost of an American restaurant is 3.12 $'s.
2  The average cost of an Italian restaurant is 2.95 $'s.
3  The average cost of a Contemporary American restaurant is 3.09 $'s.
4  The average cost of an Afghan restaurant is 2.0 $'s.
5  The average cost of a French restaurant is 3.22 $'s.
6  The average cost of a Southwest restaurant is 4.0 $'s.
7  The average cost of a Contemporary French restaurant is 2.5 $'s.
8  The average cost of a Tapas / Small Plates restaurant is 2.0 $'s.
9  The average cost of a French American restaurant is 3.0 $'s.
10 The average cost of a Seafood restaurant is 3.5 $'s.
11 The average cost of a Winery restaurant is 2.0 $'s.
12 The average cost of a Steakhouse restaurant is 3.5 $'s.
13 The average cost of a Greek restaurant is 3.0 $'s.
14 The average cost of a Fusion / Eclectic restaurant is 3.0 $'s.
15 The average cost of a Steak restaurant is 4.0 $'s.
16 The average cost of a Mediterranean restaurant is 3.0 $'s.
17 The average cost of a Vietnamese restaurant is 3.0 $'s.
18 The average cost of a Peruvian restaurant is 3.0 $'s.
19 The average cost of a Mexican restaurant is 2.0 $'s.
20 The average cost of a Sushi restaurant is 3.0 $'s.
21 The average cost of a Fish restaurant is 4.0 $'s.
22 The average cost of a Traditional French restaurant is 3.0 $'s.
23 The average cost of a Farm-to-table restaurant is 2.5 $'s.
24 The average cost of a Croatian restaurant is 2.0 $'s.
25 The average cost of a Speakeasy restaurant is 2.0 $'s.
```

menu_nypl.py

The screenshots below shows the final output of the dishprices.txt file:

menu_nypl.py M dishprices.txt U

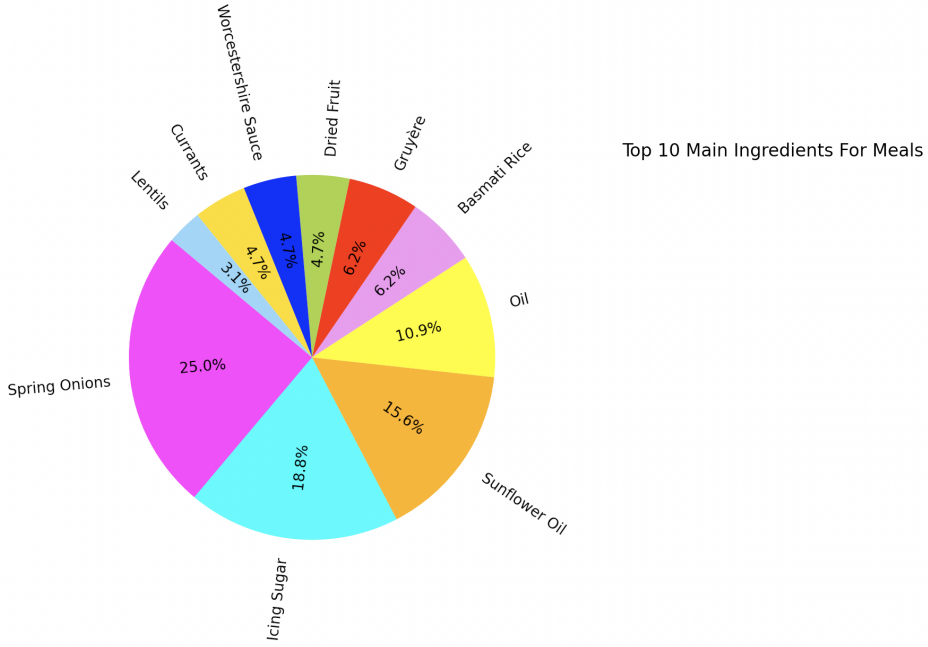
dishprices.txt

```
The Average Prices of Dishes Featured in the NYPL Menus API:
The average price of Aloxe-Corton l Er Cru, "Les Valozieres", Antonin Rodet, 1996 (Burgundy) is 75.0.
The average price of Chianti Classico Riserva, Villa Antonori 1995 (Tuscany) is 30.5.
The average price of Pinot Grigio, Peter Zemmer 1997 (Alto Adige) is 29.0.
The average price of Pinot Grigio, J. Tiefenbrunner 1996 DOC (Trentino) is 28.0.
The average price of VEAL CHOP PARMIGIANA (house specialty) is 26.75.
The average price of VEAL CHOP red peppers & rosemary potatoes is 25.75.
The average price of GRILLED LARGE SHRIMP with vegetable risotto is 24.95.
The average price of GRILLED ATLANTIC SALMON grilled summer vegetables is 22.75.
The average price of VEAL MARSALA portobellini mushrooms and rosemary potatoes is 22.5.
The average price of ASSORTED ITALIAN SAUSAGES & POLENTA tomato & parmigiano gratine is 18.95.
The average price of RIGATONI eggplant, portobellini mushrooms, tomato & ricotta sarda is 17.95.
The average price of Seafood Antipasto any six seafood items is 17.95.
The average price of TORTELLONI chicken & porcini filled, marscapone & truffle sauce is 17.95.
The average price of GNOCCHI CON FEGATINI DI POLLO chicken livers is 17.75.
The average price of LINGUINI ALLA BOLOGNESE veal, portobellini & cremini mushrooms is 17.5.
The average price of FETTUCINE AL FRESCO Bucheron goat cheese, fresh tomato & basil is 16.95.
The average price of Mixed Antipasto any three seafood & four vegetables is 16.95.
The average price of CALABRIA PEPPERONI mozzarella & tomatoes is 16.5.
The average price of MARGHERITA tomatoes, mozzarella & basil is 15.75.
The average price of MUSSELS TOSCANE fresh tomatoes is 12.5.
The average price of FRIED ARTICHOKES ALLA GIUDEA is 9.5.
The average price of Cabernet, Pedroncelli is 7.95.
The average price of Chardonnay, Glass Mountain is 7.95.
The average price of Merlot, Madame Costeau is 7.95.
The average price of Zinfandel, Rabbit Ridge is 7.95.
The average price of Vernaccia,Mormoraia is 7.75.
The average price of Gavi di Gavi Barrique, La Guardia is 7.5.
The average price of Chardonnay, Lindemans Bin 65 is 6.5.
The average price of Montepulciano, Masciarelli is 6.5.
The average price of FRIED ARTICHOKES is 5.95.
The average price of ROASTED ROSEMARY POTATOES is 5.95.
The average price of Guinea chicken saute, fresh mushrooms is 1.0.
The average price of Spring chicken fricassee, family style is 0.9.
The average price of Broiled pork tenderloin, fried parsnips is 0.7.
The average price of Broiled sweetbreads, Infante is 0.7.
The average price of Fried frogs' legs, sauce tartare  is 0.7.
The average price of Roast baby lamb with mint sauce is 0.7.
The average price of Broiled green bluefish, Julienne potatoes  is 0.65.
The average price of Boiled Kennebec salmon, Magenta is 0.6.
The average price of Broiled weakfish, pommes Duchesse is 0.6.
The average price of Filet of flounder poached, Martinique is 0.6.
The average price of Puree of white beans, croutons soufflee is 0.25.
```

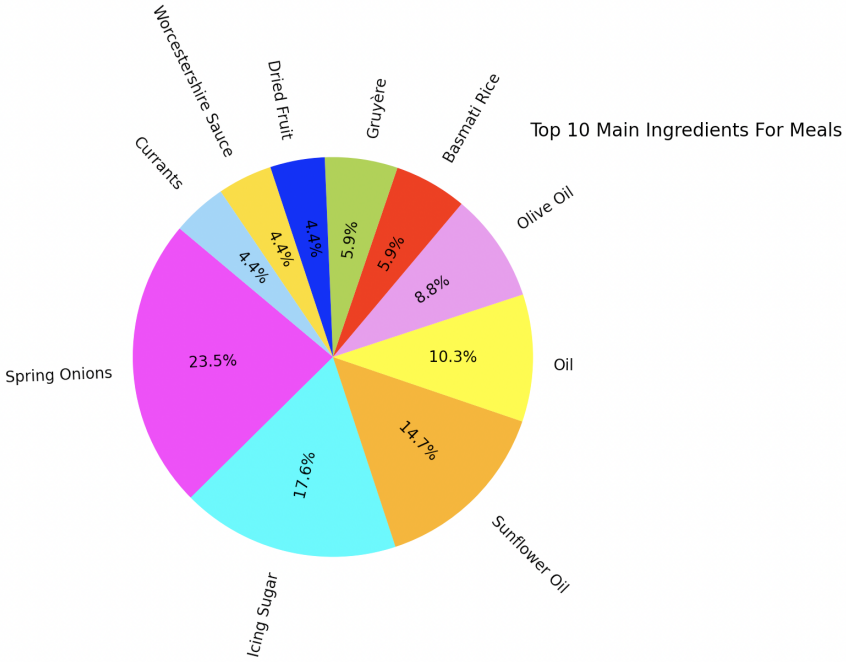
5. **The visualization that you created (i.e. screen shot or image file)**

MealsAPIFinal.py:

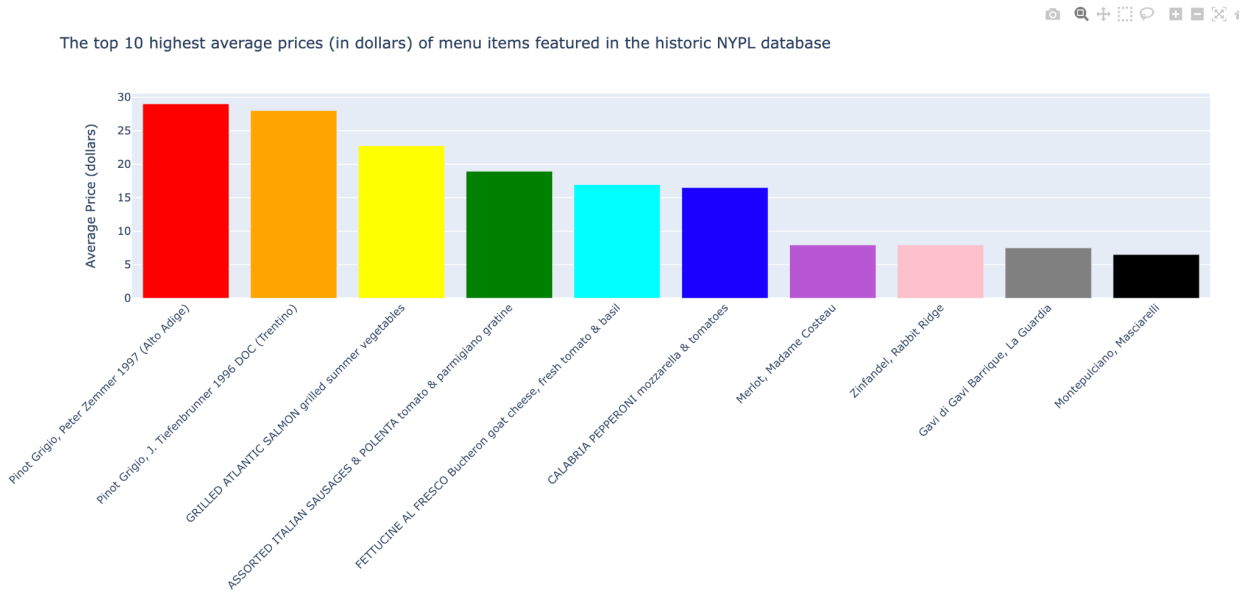
The visualization below shows the Top 10 Main Ingredients for meals in the database, taken from The Meals DB API. The goal of this pie chart is to help students easily see 10 main ingredients that are common in meals (all of which are in the pie chart) in order to help guide their grocery shopping. This visualization also helps a student identify which main ingredients are



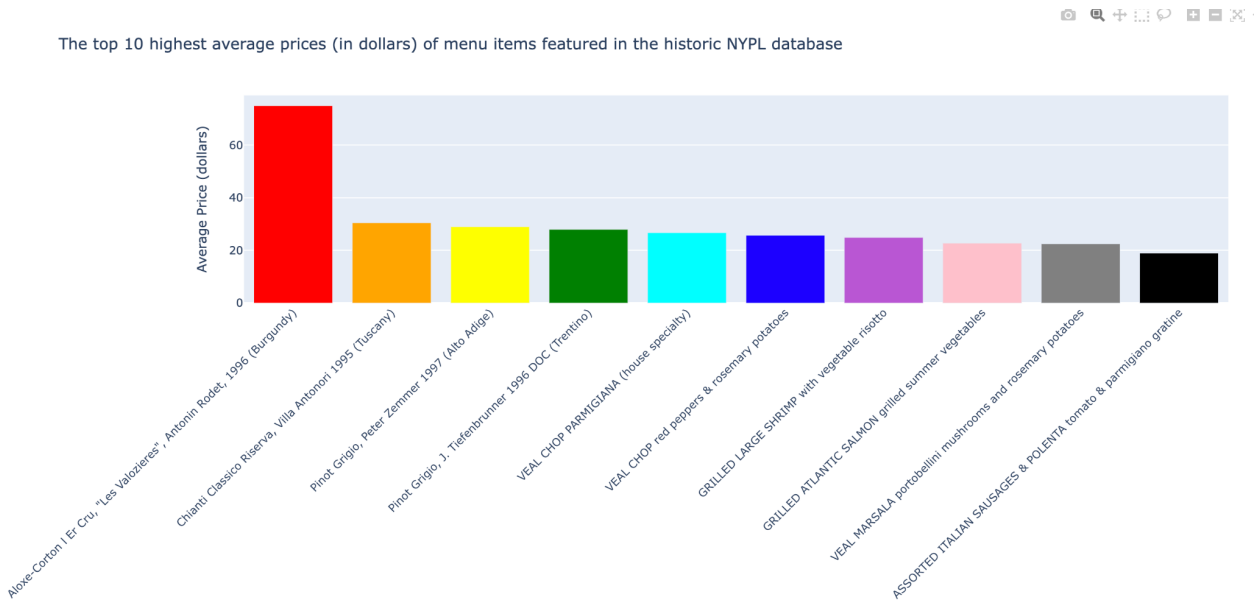
Below is an updated version of the Top 10 Main Ingredients, after 25 more meals have been added to the database, thereby updating which ingredients are the top 10 for the meals in the database.



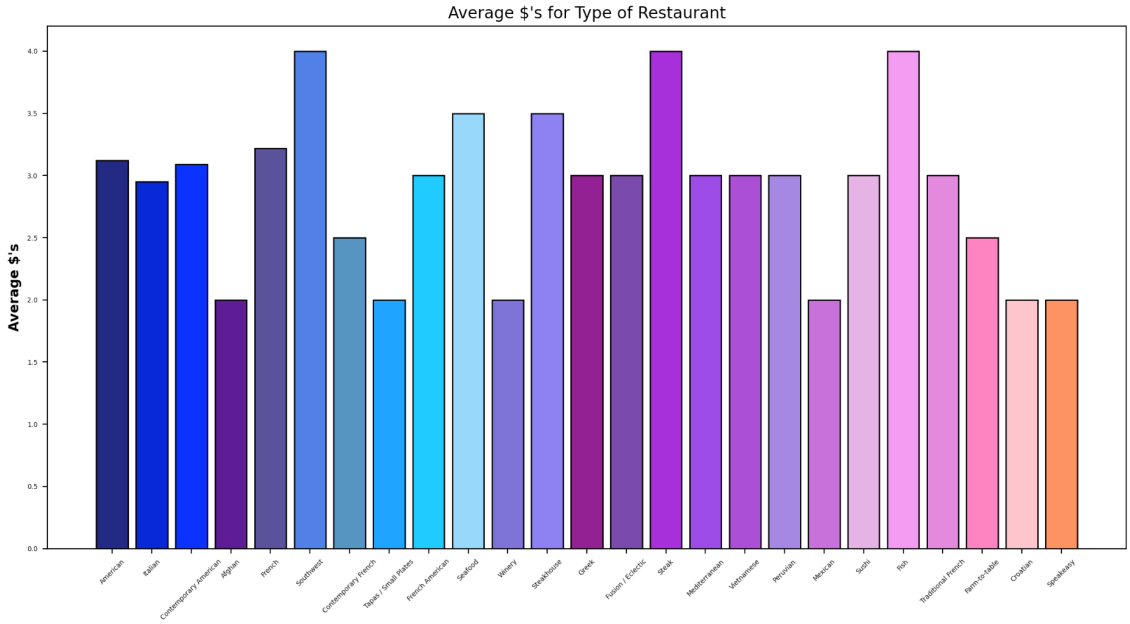
The visualization below shows the top ten highest prices of the first 25 dishes added into the Menus API database:



Once run four times, this visualization shows the top ten highest prices among the 100 total dishes in the menus API database:



The visualization below shows the average $'s for each type of restaurant from the Top100 Diners website:



6. Instructions to run code for each website/API:

First, ensure that Visual Studio Code, SQLite Database Browser, Plotly and Matplotlib are fully installed on your computer. The API The Meals DB does not need to be pulled up or accessed before running the code; however, the What's On the Menu API does need to be accessed in order to receive an API key. Next, open up the zipped file. The files you should open and run (in no particular order) are: MealsAPIFinal.py, 100Diners.py, menu_nypl.py. With SQLite DataBase Browser installed, the tables within the database will be viewable.

MealsAPIFinal.py:
Each time the code is run, 25 new meals are added to the meals table. These meals are generated by selecting a randomized ingredient from the Ingredients table (which stores all of the possible ingredients found on The Meals DB API). Using the selected random ingredient, the code adds all of the meals that contain the specified ingredient as its main ingredient to the table. The main ingredient id for each of these meals matches up with the id of the main ingredient in the ingredients table. Once the desired amount of meals are inside the Meals table, calculations are run in order to calculate which ingredient is the main ingredient for the highest number of meals, and which ingredient is the main ingredient for the lowest number of meals. These two ingredients are written to the csv file, along with the main ingredients with the top 10 highest meals, and the counts of meals that each has. Our intention was to produce these numbers for a college student interested in finding meal inspiration for cooking their own meals. If they are provided with an ingredient that has the highest number of meals (that use that ingredient), they can use this information to aid their grocery shopping. One should pick ingredients with high numbers of meals so that the ingredient can be utilized several times and/or for different meals in order to create variety in the student's diet. Similarly, if an ingredient has a low number of meals (that use the ingredient as a main ingredient), the student may want to avoid buying the ingredient when grocery shopping. A pie chart will be displayed representing which ingredients are the top 10 main ingredients (meaning they are the main ingredient for the highest number of meals) out of the meals in the Meals table in the database.

100Diners.py
Each time the code is run, 25 new restaurants are added to the restaurants table. From the Top100 Diner website, the restaurant, type of restaurant, price in dollar signs, reviews, location, and an id for each were displayed in the database. With this information, we calculated the averages of the amount of dollar signs for each type of restaurant. This information was added into a .txt file. The next section of the code shows this information in a bar graph. Once the code runs four times and the database is complete, the final graph that shows is finalized. Types of restaurants are portrayed on the x-axis and $ averages are portrayed on the y-axis.

menu_nypl.py
When this code is run, 25 new dishes, as well as values pertaining to it, are added to the Dishes table. Each time it is run, an additional 25 new dishes are added to the table and a visual pops up conveying the top 10 most expensive price averages in the database. This occurs 4 times, until a total of 100 dishes have been added. Once the database is full with the data, the data visualization conveyed reveals the total top 10 most expensive price averages that each dish cost when featured on a menu. It is important to consider the vast range of years that the menus were created, ranging from hundreds of years ago to just a singular number of years. Resultantly, there are some dishes that seem significantly *more* or *less* expensive than they typically would be in our current era. It can be interesting to see these trends found in the bar graph, as one can compare their expectations with what was found in the database.

At the end, there should be 100 rows in each table of the following: Meals, Dishes, and Diners.
The code will create 3 CSV files named: Meals_Calculations.txt, restuarant_calculations.txt, and dishprices.txt, that can be opened in Google Sheets or Excel that shows our three calculations (these are also linked above).

7. Documentation for each function that you wrote. This includes the input and output for each function (screenshots of docstrings)

The following show the documentation for the functions used when retrieving data from **MealsAPIFinal.py**:

```
def setUpDatabase(db_name):
    """
    This function creates the database titled 'Meals2.db' that will be referenced.
    It returns the connection to the database and the cursor.
    """
```

```
def setUpIngredientsTable(data, cur, conn):
    """
    This function creates the table Ingredients in the database.
    It takes in 3 arguments: data (such as the ingredients list retrieved from the API),
    the database cursor, and database connection object.
    The table has 2 columns: id and Ingredient.
    The id is autoincremented and the Ingredients are taken from
    a list of all of the ingredients on the 'themealdb.com'.
    This function does not return anything
    """
```

```
def create_meals_tables(cur, conn):
    """
    This function creates the table Meals in the database.
    It takes in 2 arguments: the database cursor, and database connection object.
    It does not return anything.
    """
```

```
def update_meals_table(cur, conn):
    """
    This function updates the Meals table. It takes in 2 arguments:
    the database cursor, and database connection object.
    It calls the find_meals function on each of the ingredients in the Ingredients table.
    It then adds the meals to the table with a specification of which main ingredient is used,
    with the Main_ingredient_id. This function does not return anything.
    """
```

```
def num_meals_for_ingredient(cur, conn):
    """
    This function takes in two arguments:
    the database cursor, and database connection object.
    It then counts the number of meals each ingredient is the main ingredient for.
    It returns a list of tuples, each of which contains the ingredient and the count of meals it is the main ingredient for.
    This function calls on both tables Ingredients and Meals and joins them in order to retrieve this information.

    """
```

```
def top_ten(lst_tups):
    """
    This function takes in a list of tuples (called lst_tups, i.e. the
    one that is returned after running the num_meals_for_ingredients() function).
    The function then sorts this list.
    It returns the 10 Ingredients with the most meals.
    If there are 10 or more meals in the database, it will return the top 10.
    Otherwise (if, for example, 25 meals are added to the database but many share
    the same main ingredient and therefore 10 have not been added yet), the function
    will return the ingredients that are currently in the database. """
```

```
def write_csv(tup, top_10, filename):
    """
    This function takes in a tuple, a list tuples, and a filename.
    The tuple should contain the ingredient with the most meals, followed by the
    ingredient with the least. The list of tuples is called top_10 (i.e. the
    one that is returned after running the lst_top_10() function) contains the
    10 Ingredients with the highest number of meals.
    The function first writes a title for the file.
    It then writes the tuple of the ingredients with the highest and lowest counts of meals.
    Finally, it writes the Top 10 Ingredients along with their counts of meals.
    This data is all written to a csv file, and saves it to the passed filename.
    This function does not return anything.
    """
```

```
def main():
    """
    This function first sets up the database, which is named Meals2.db.
    It then created the Ingredients table, accessing the Meals DB API.
    """
```

The following show the documentation for the functions used when retrieving data from **100Diners.py**:

```
def get_data_from_website(url):
    """
    This function scrapes the data from the Top100 Diners website using
    BeautifulSoup. It returns multiple lists of the names, the type,
    the dollar signs (cost), the reviews, and the location for each restaurant.
    """
```

```
def setUpDatabase(db_name):
    """
    This function creates the database titled "restaurants_database".
    It returns the connection to the database and the cursor.
    """
def create_table(cur,conn):
    """
    This function creates the restaurants table in the databse.
    It takes 2 arguments such as the database cursor, and database connection
    object. It does not return anything.
    """
```

```
def setup_restaurantstable(data, cur, conn):
    """
    This function puts the data into the restaurants_table.
    It takes in 3 arguments such as data, which is the url of the website,
    the database cursor, and the database connection object.
    The table has 6 columns. They are id, restaurant name, type of restaurant,
    cost of restaurant (in $'s), number of reviews, and location.
    """
```

```
def calculate_average(cur, conn):
    """
    This function calculates the average number of dollar signs for each type of
    restaurant. It takes in 2 arguments such as the database cursor and
    the database connection object. It then wrties these averages in the
    restaurant_calculations.txt file. The last section of this function creates
    a visual using matplotlib to create a bar graph of these adverages.
    """
```

The following show the documentation for the functions used when retrieving data from **menu_nypl.py**:

```
def setUpDatabase(db_name):
    """
    This function sets up the database 'menus.db' that will be accessed with later functions.
    It returns the connection to the database and the cursor.
    """
```

```
def getDishInfo():
    """
    This function accesses the Menus API from NYPL and extracts a list of dictionaries containing menu items and information pertaining to the
    It uses the token that was given to us after requesting one.
    With the data extracted, the function then creates and returns a dictionary containing the dishes and the information.
    """
```

```
def create_dish_table(cur, conn):
    """
    This function sets up the database with the dictionary returned by the getDishInfo function.
    It creates the table Dishes and, in increments of 25, inputs the dishes and a selection of data per dish.
    This function does not return anything, but rather inputs data into the database after each time it is run.
    Once run 4 times, all of the data from the data dictionary will be displayed on the database.
    """
```

```
def calculate_avg_price(cur, conn):
    """
    This function selects the dish name and the highest and lowest prices it has been listed for from the database.
    With this data, the average price is calculated by adding the two prices together and dividing it by two, then creating a tuple with the
    This function returns a sorted list of the tuples in descending order.
    """
```

```
def write_calculation(cur, conn):
    """
    This function writes the dish names and average prices to a csv file.
    This function does not return anything.
    """
```

```
def plot_avg_prices(cur, conn):
    """
    This function plots the top 10 highest average prices from what the list returned by the calculate_avg_price function.
    The graph is a bar graph with the dish names on the x-axis and the average prices on the y-axis. Each value has a color in rainbow order.
    As 25 additional items are added to the database when the function is run repeatedly (for four times), the bar graph updates to display th
    When run, this function displays the bar graph.
    """
```

```
def main():
    """
    This function calls all of the above functions.
    It sets up the 'menus.db' database, creates the Dishes table in the database, calculates the average dish prices, writes these calculatin
    """
```

8. Documentation of Resources:

Date	Issue Description	Location of Resource	Result (did it solve the issue)?
11/18	Kept getting warning messages about creating a connection over HTTPS	https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warningswarnings.warn	Yes, this source showed how to import 'urllib3' to assist in connecting to the server in order to extract the data without a warning message printed
11/19	Code limits 25 items to be added to the database; however, when it is run, the 25 items are replaced with 25 new items instead of being inserted in addition to the previous items in the database.	Office Hours (Xinghui)	Yes, figured out how to successfully add 25 items into the database each time the code is run, implementing a counter variable and adding the counter to the amount of rows currently in the table in each INSERT statement.
11/22	The Fruityvice API does not have enough information in order to add at least 100 items to the table (only information about 23 fruits). Therefore, this API does not fit with our original project plan or the requirements for the project.	We located a new API to use (http://nypl.github.io/menus-api/), through searching through the "Food and Drink" section in the link provided to us for free APIS: https://github.com/public-apis/public-apis#food--drink.	The newly selected API proved to produce the desired amount of information and fit the requirements of the project.

11/30	The percentages shown in the pie chart were overlapping for some items in the pie chart, thereby making the visual difficult to read.	Researched how to rotate these labels https://stackoverflow.com/questions/64411633/how-to-rotate-the-percentage-label-in-a-pie-chart-to-match-the-category-label-ro	The site helped provide guidance on how to correctly rotate these percentages, thus making the pie chart easier to read.
12/1	Had trouble limiting the data input to only 25 items each time the code was run	Dr. Ericson	She helped debug the code and figure out how to manipulate it to load 25 items each time the code was run
12/2	Wasn't sure what the available colors were for plotly	https://community.plotly.com/t/plotly-colours-list/11730/3	This link showed a large list of the colors available with plotly, allowing me to easily select 10 colors from the list