Stephen Fedewa, Travis Simmons, David Forystek

Team: xX\_nohaxjustFoodQuest\_Xx

Github repository: <https://github.com/TheNerdyGinger/SI-206-Final-Project>

Presentation: <https://docs.google.com/presentation/d/1LI6HW95cODhf26zRZhB2vRyzmFVePZMW0FBYBj2CMgQ/edit#slide=id.p>

SI 206 Final Project Report: Foodquest

1. The Goals for your project

* We wanted to take a recipe and find the healthiest brands from around the world to use as ingredients. This is an ideal scenario where we can easily get any food item we want regardless of location. (Maybe teleportation has already been invented).

1. The goals that were achieved

* Could take the first 10 brands from a search query and find the healthiest option among them.
* Found specific brand options for most ingredients we searched for.

1. The problems that you faced

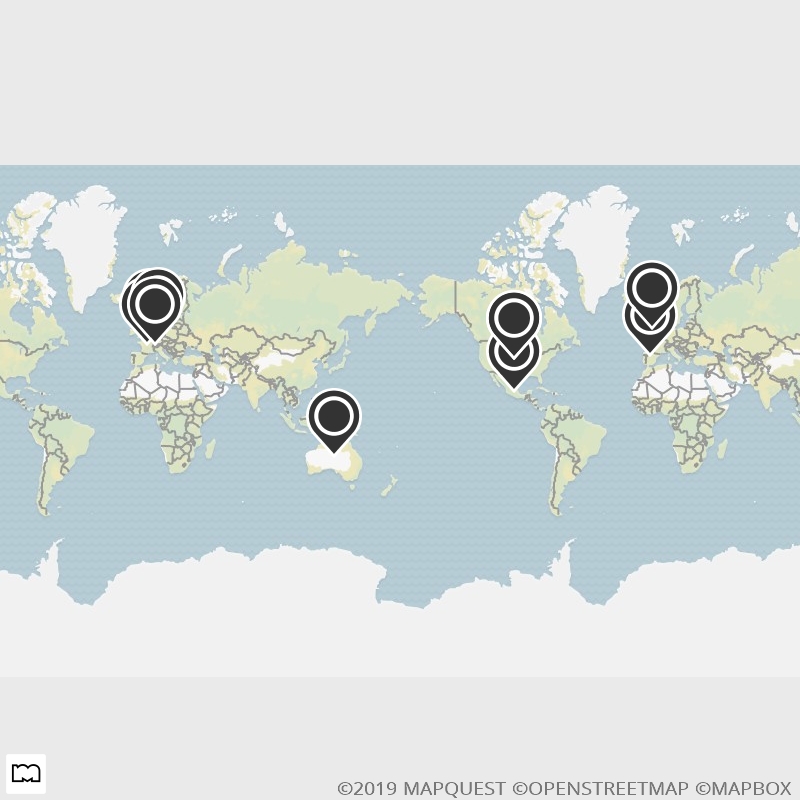
* We found that we couldn’t search for ingredients by product name. For example, if a recipe called for “black pepper”, searching by product could come up with “Black Pepper Potato Chips.” We had to switch to searching by category.
* Searching by category worked most of the time. Most basic ingredients had categories associated with them. However, certain ones did not. This wasn’t helped when recipes had weird naming conventions such as “sugar (granulated).” To solve this, we had to come up with some failsafe methods in case we didn’t find any specific brands.

1. Your file that contains the calculations from the data in the database

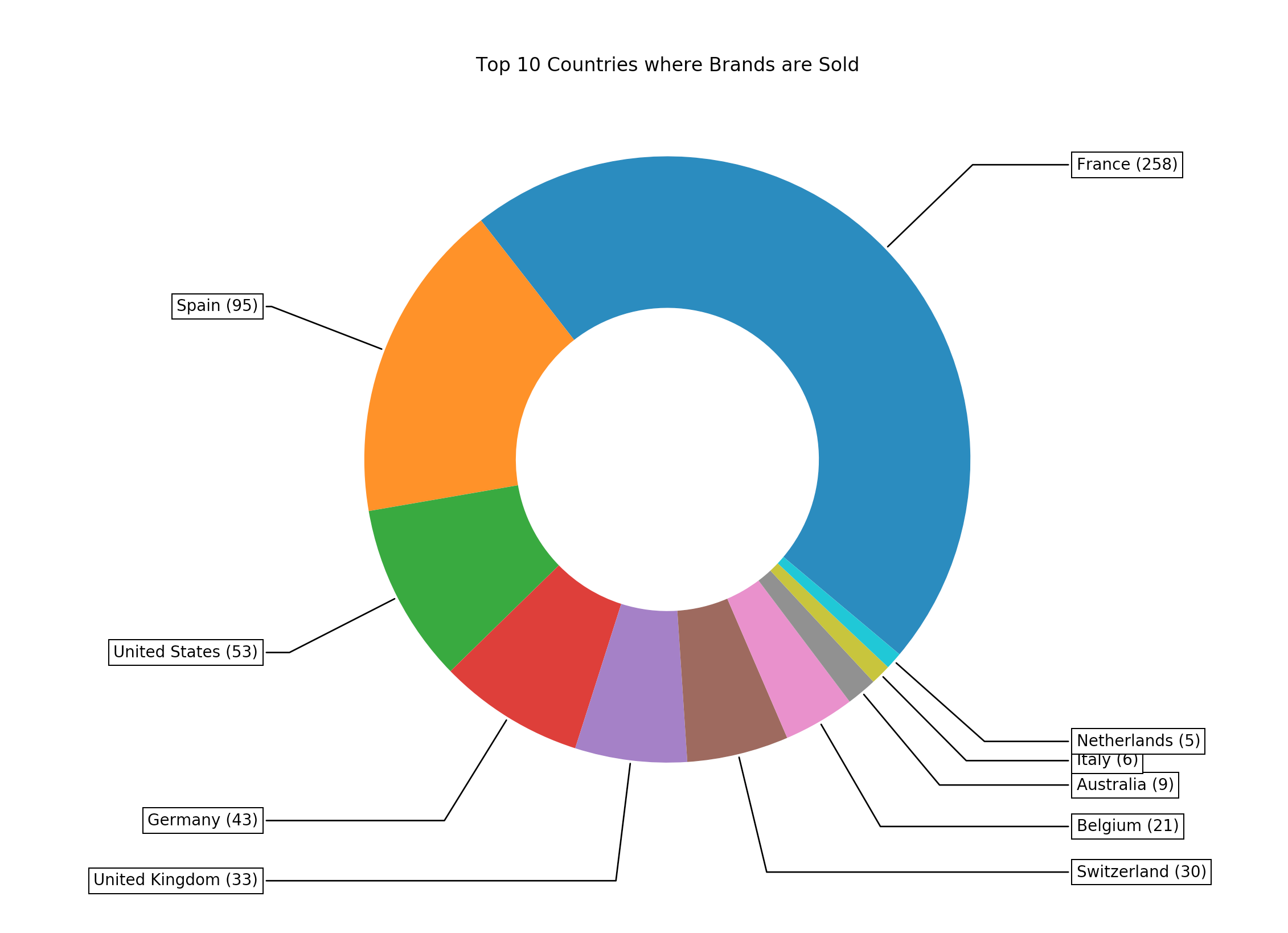
* Foodquest.db database, recipe\_output.txt for output.

1. The visualizations you created (screenshots)

Map of the most common countries to show up in our brand data. Made using the mapquest API



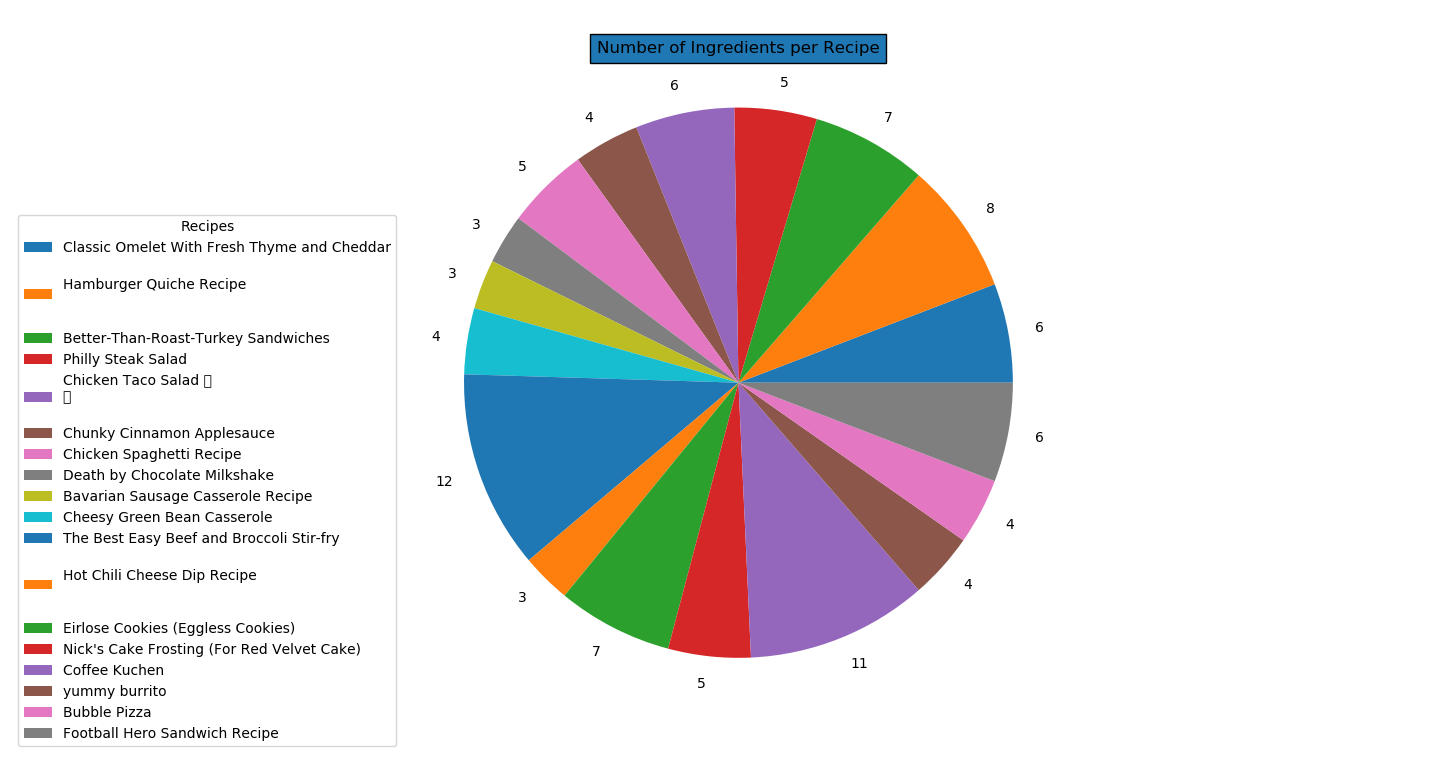
Pie chart showing the 10 most common countries brands were sold in



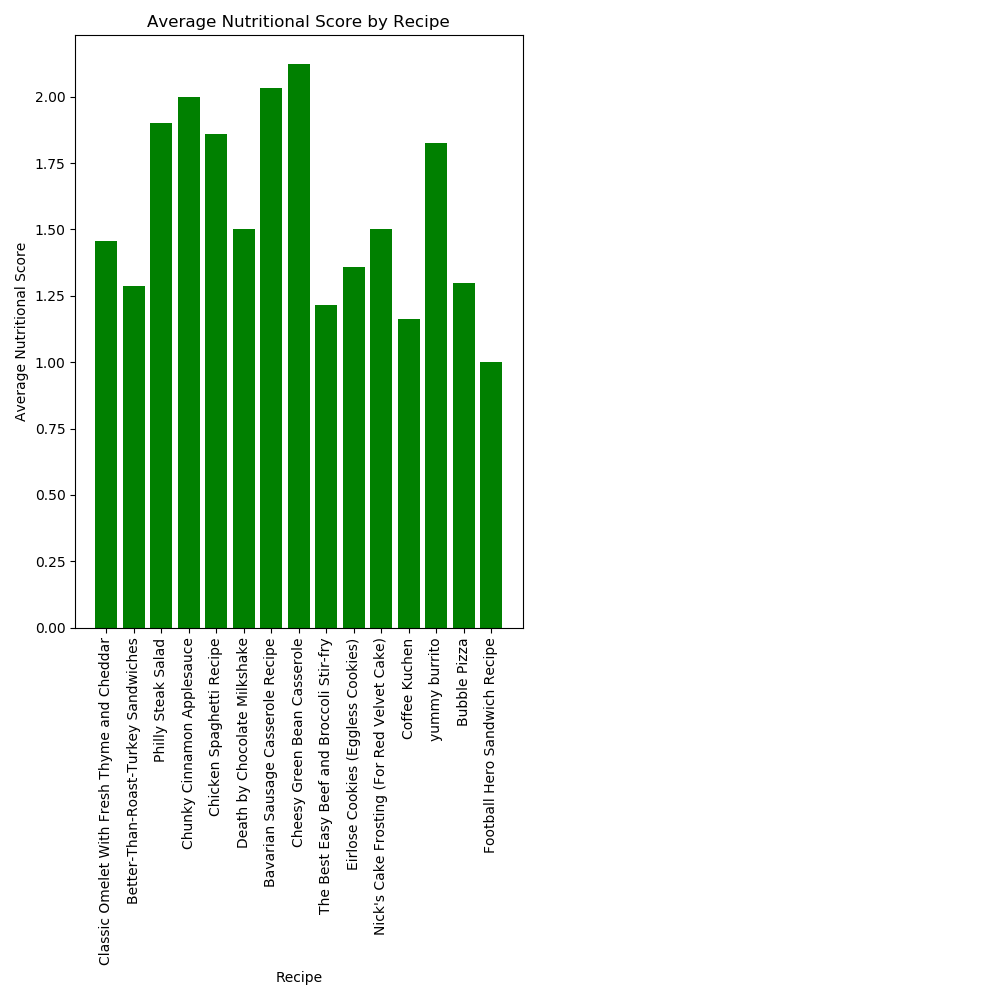
Word Cloud showing how common each ingredient was in each recipe. The larger the text, the more recipes this ingredient showed up in. Made using Plotly



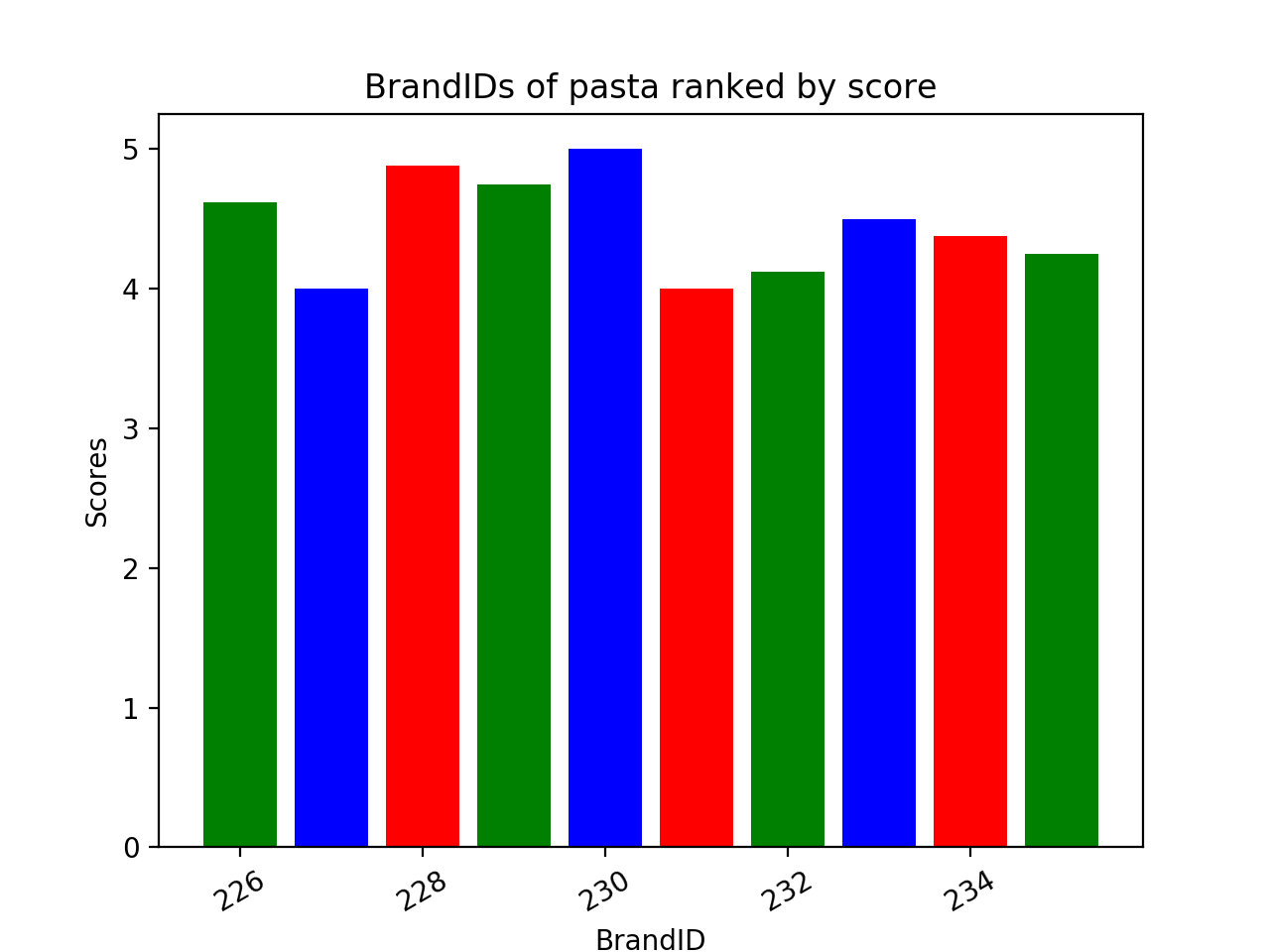
Pie chart showing the number of ingredients in each recipe. Made using Matplotlib



Bar graph showing the average nutrition score for each recipe. Made using MatPlotLib



Bar graph showing all of the brands for the “pasta” ingredient and their scores after breaking ties with their distances from Ann Arbor. Made with Matplotlib



1. Instructions for running your code
   1. Install the following libraries:

* Geopy <https://geopy.readthedocs.io/en/stable/#installation>
* Inflect <https://pypi.org/project/inflect/>
* Plotly <https://plot.ly/python/getting-started/>
  1. If starting from the turned-in zip file, delete foodquest.db or erase all records from it. Open off\_bookmark.txt, erase all text, write ‘0’, and save the file.
  2. Open mapquest.py and keep running it until it prints “No more countries to input” (Run about 12 times)
  3. Run recipe\_puppy.py until it prints “NO NEW RECIPES” (About 18 times)
  4. Run off.py until it prints “No more ingredients to check” (About 66 times)
  5. Run combine\_data.py until it prints “No more ingredients to check” (About 66 times)
  6. Run recipes\_and\_best\_brands until it prints “No more recipes to check” (About 18 times)
  7. Run visualizations.py once
  8. View the text output in recipe\_output.txt and the visualization files.

1. Documentation for each function that you wrote. Includes the input and output for each function

**Mapquest.py:**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database name. Returns cur and conn

def get\_countries(filename):

* Takes in a CSV final with a list of countries as input, returns a python list of countries as output.

def get\_mapquest(country\_list, cur, conn, start):

* Takes in the country list from get\_countries, database cursor, database connection, and the start variable as input.
* Calls mapquest geocode API to receive latitude and longitude for each country in country\_list and stores in the database.
* Start variable used to limit commits to the database to 20 per run of code.
* Does not return anything.

def get\_distance(cur, conn, start):

* Takes in the database cursor, database connection, and start variable as input
* Uses mapquest geocode API to fetch latitude and longitude of Ann Arbor, MI
* Fetches latitude and longitude for every country in country\_list and uses geopy library to calculate distance. Start variable used to fetch 20 entries from the database at a time.
* Commits distance calculation to database at 20 entries at a time.
* Does not return anything.

def main():

* sets database to “foodquest.db”, sets database cursor and database connection
* Sets up the Countries, Lat\_lng, and Distances tables
* calls country\_list, get\_mapquest, and get\_distance
* Selects countries from database at 20 per run of code

**Recipepuppy.py**

def get\_data(ingredients, recipe):

* Requests data from the RecipePuppy API and returns a json dictionary of the recipes and ingredients

def read\_file(filename):

* Reads the file that contains multiple parameters for the get\_data function and returns each line in the file

def create\_tuples(filename):

* Returns a list of tuples with the first element being the recipe name and the second element being the list of ingredients for the recipe

def create\_database(filename):

* Takes the data taken by create\_tuples and creates tables in the database that contain the following:
  + A table with recipes and unique recipe IDs
  + A table with ingredients and unique ingredients IDs
  + A table matching each unique ingredient ID with a recipe ID

**Off.py**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database name. Returns cur and conn

def format\_ingredient\_name(ing):

* Takes in an ingredient name as a string, converts it to lowercase, replaces spaces with hyphens, and uses inflect library to pluralize it. Returns the newly formatted ing.

def format\_ingredient\_no\_plural(ing):

* Takes in an ingredient name as a string, converts it to lowercase, and replaces spaces with hyphens. Returns the newly formatted ing.

def format\_countries(countries, cur, conn):

* Takes in a list of country names, and the cur and conn of the database
* For each country in country names:
  + If already in the database, get its country ID and append it to a list of country ids called c\_ids.
  + Else (if not found), make a mapquest request just like mapquest.py to add that country and its necessary data to the database. Appends the newly inserted country\_id to c\_ids.
* Return c\_ids

def insert\_into\_tables(cur, conn, product\_name, nutriscore, countries, ingredient\_id)

* Takes in the database’s cur and conn, a product name, it’s nutrition score, a list of countries where the product is sold, and its ingredient id
* Determines a brand\_id and inserts brand\_id, product\_name, ingredient\_id, and nutriscore into Brands\_and\_scores table
* For each c\_id in countries:
  + Insert brand\_id and c\_id into Brand\_countries\_sold table

def insert\_if\_not\_found(cur, conn, ing\_id):

* Takes in the cur and conn of the database and an ingredient id.
* This function is used if a specific brand could not be found for a given ingredient.
* Determines a brand\_id and inserts brand\_id, the generic ingredient name, and a nutriscore of 0 into Brands\_and\_scores table
* Inserts brand\_id and a country id of 225 (United States) into Brand\_countries\_sold table

def parse\_json(data, cur, conn, ingredient\_id):

* Takes in a python list of json data on potential brands, the database cur and conn, and an ingredient id
* Loops until we’ve inserted 10 brands into the database, or until we’ve checked every product in the json data.
* For each brand:
  + Find nutriscore
  + If nutriscore not found
    - Find nova\_group
    - If nova\_group not found
      * Restart loop, move to the next brand
  + Else, convert nova\_group to a proper nutrition score
  + If nutriscore is a string (character), convert it to its corresponding integer score
  + Obtain product name and list of countries. Convert countries to country\_ids by calling format\_countries(countries, cur, conn)
  + Call insert\_into\_tables(cur, conn, product\_name, nutriscore, countries, ingredient\_id)
* If no brands were inserted, call insert\_if\_not\_found(cur, conn, ingredient\_id)

def request\_url(cur, conn):

* Takes in the cur and conn of the database.
* Creates Brands\_and\_scores and Brand\_countries\_sold tables if they don’t exist
* Uses a number from off\_bookmark.txt to determine where to begin inserting into the tables. (integer variable pos) And to check if we’ve already inserted everything we need to.
* Selects one ingredient to look at and calls format\_ingredient\_name to pluralize it.
* Tries to make a request to the Open Food Facts API, searching by category using the pluralized ingredient name. Loads the result into a list called data. Prints “error reading from url” and makes data an empty list if there is an exception.
* If no products are found:
  + Tries to make another request, this time with a non-pluralized version of the ingredient name. Loads response into list data2 Prints “error reading from url” and makes data2 an empty list if there is an exception.
  + If still no products found: call insert\_if\_not\_found(cur, conn, pos)
  + Else, call parse\_json(data2, cur, conn, pos)
* Else, call parse\_json(data, cur, conn, pos)
* Closes off\_bookmark.txt, and reopens it to write pos+1 into it. Closes bookmark.txt

def main():

* Calls setUpDatabase(“foodquest.db”) which returns cur and conn variables
* Calls request\_url(cur,conn) to begin collecting data.

**Combine\_data.py**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database

def get\_shortest\_distance(tup, cur, distances):

* Takes in a tuple containing a brand\_id, the database cur, and a list of distances from Ann Arbor for a few countries
* Grabs which countries the brand with the passed brand\_id is sold in, and uses the distances list to determine those countries’ distance from ann arbor.
* Loops through the returned list of countries to find which one has the shortest distance.
* Returns the float value of the shortest distance.

def insert\_combined\_score(start, brands\_and\_scores, cur, conn):

* Takes in an index of insertion representing an ingredient id, a list of brands and their combined scores, and the cur and conn of the database
* for b in brands\_and\_scores:
  + Insert (start, b[0], b[1]) into Combined\_score table

def insert\_best\_brand(start, brand, cur, conn):

* Takes in an index of insertion representing an ingredient id, a brand id, and the cur and conn of the database.
* Inserts (start,brand) into Best\_brands table

def combine(cur, conn):

* Takes in the database cur and conn.
* Creates Combined\_score and Best\_brands tables if they don’t exist.
* Determines the position to start inserting and checks if we’ve already checked all of the data.
* Grabs all the brands and nutrition scores for one ingredient and saves them into a list of tuples called brands\_and\_scores
* Grabs all of the unique ids for countries sold for all the brands in brands\_and\_scores. (If multiple brands are sold in france, france will still show up once in the list)
* Makes a dictionary of distances for all of the countries in countries list.
* Sorts brands\_and\_scores by nutrition score in descending order, and determines the highest score.
* For every brand that ties for the highest score:
  + Call get\_shortest\_distance to find the closest country the brand is sold in. (These will be used as tie breakers)
  + Adds this shortest distance as a 3rd element for those tuples.
* Sort tied brands by distance
* Determines subtractor variable = 1/(number of tied brands)
* Count = 0
* For each tied brand:
  + Subtract (count \* subtractor) from score (tuple[1])
* Call insert\_combined\_score(start, brands\_and\_scores, cur, conn) and insert\_best\_brand(start, brands\_and\_scores[0][0], cur, conn)

def main():

* Calls setUpDatabase to get cur and conn, then calls combine(cur, conn)

**Recipes\_with\_best\_brands.py**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database

def main():

* Calls setUpDatabase to get cur and conn.
* Creates Recipes\_and\_best\_brands table if it doesn’t exist
* Determines the point to ‘start’ inserting and checks if we’ve already run through each recipe.
* Grabs all of the ingredients for one recipe from the database. Saves them in list ‘ings’
* For i in ings:
  + Find the best brand for ingredient i from the best\_brands table
  + Insert (start, best\_brand) into Recipes\_and\_best\_brands table.

**Recipe\_output.py:**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database

def join\_databases(cur, conn, filename):

* Takes in the database cur and conn and a file name to output to.
* Performs a join on Recipes, Brands\_and\_scores, and Recipes\_and\_best\_brands tables to create a list of tuples containing recipe names and best\_brand names instead of ingredients. Saves the data in output\_table list variable
* Opens a file with the given filename to write to.
* Loops over output\_table to output the data to a text file in the following format:
  + *Recipe\_name1: brand1, brand2, … \n*
  + *Recipe\_name2: brand1, brand2, … \n*
* Closes the output file and returns output\_table

def main():

* Calls setUpDatabase to get cur and conn.
* Calls join\_databases(cur, conn, "recipe\_output.txt")

**Visualizations.py**

def setUpDatabase(db\_name):

* Takes in a db\_name as a string and sets up cur and conn with the given database name. Returns cur and conn

def grab\_wordcloud\_data(cur, conn):

* Takes database cursor and database connection as input.
* Fetches ingredients from database
* Adds all the ingredients to a string, returns string of ingredients as output

def word\_cloud(word\_string):

* Takes the word string from grab\_wordcloud\_data as input
* Uses plotly to generate word cloud
* Does not return anything

def grab\_pie\_data(cur, conn):

* Takes database cursor and database connection as input
* Fetches countries from database and appends to list
* Returns list of countries as output

def pie\_graph(country\_list):

* Takes in list of countries from grab\_pie\_data as input
* Sorts countries by number of brands sold
* Generates pie graph
* returns list of sorted countries as output

def grab\_staticmap\_data(cur, conn, country\_list):

* Takes in database cursor, database connection, sorted country list returned by pie\_graph as input
* Fetches coordinates of all those countries from database and appends to list
* Returns list of coordinates as output

def staticmap(coord\_list):

* Takes in list of coordinates from grab\_staticmap\_data as input
* Calls mapquest static map API to produce a static map of the top countries where brands are sold marked on the map
* Uses webbrowser module to open static map in web browser
* Does not return anything

def grab\_barchart\_data(cur, conn):

* Takes in database cursor and database connection as input
* Fetches ingredients and recipes from database
* Calculates average nutrition score for each recipe
* Produces bar graphs with ingredient/recipe on the x-axis and nutrition/average score on y-axis
* returns dictionaries as ingredient\_score\_dict and recipe\_score\_dict

def barchart(ingredient\_score\_dict, recipe\_score\_dict):

* Takes ingredient\_score\_dict and recipe\_score\_dict from grab\_barchart\_data as input
* Produces bar graphs with ingredient/recipe on the x-axis and nutrition/average score on y-axis
* Does not return anything

def get\_data(dbname, command):

* Sets up connection to database
* Returns data from database

def plot\_bar\_graph():

* Plots a bar graph containing the nutrition scores for all brands of pasta sorted by BrandID

def plot\_pie\_chart():

* Plots a pie chart showing the number of ingredients for each recipe in the database

def main():

* Calls all of the previous functions to produce visualizations

1. Document all resources you used in the following format

Date: Issue Description: Location of Resource: Result (Did it solve the issue?):

|  |  |  |  |
| --- | --- | --- | --- |
| DATE | ISSUE | RESOURCE | RESULT |
| 12/11/19 | Joining 3 tables at once | <https://www.dofactory.com/sql/join> | solved |
| ~12/1/19 | Selecting max id from table. | <https://stackoverflow.com/questions/7604893/how-do-i-select-an-entire-row-which-has-the-largest-id-in-the-table> | solved |
| 12/15/19 | Capitalizing every word in a string. | <https://stackoverflow.com/questions/1549641/how-to-capitalize-the-first-letter-of-each-word-in-a-string> | solved |
| 12/15/19 | Adding a legend to a pie chart | <https://stackoverflow.com/questions/19852215/how-to-add-a-legend-to-matplotlib-pie-chart> | solved |
| 12/14/19 | Opening a url in python | https://stackoverflow.com/questions/4302027/how-to-open-a-url-in-python | solved |
| 12/14/19 | Creating Donut Pie Graph | https://matplotlib.org/3.1.1/gallery/pie\_and\_polar\_charts/pie\_and\_donut\_labels.html#sphx-glr-gallery-pie-and-polar-charts-pie-and-donut-labels-py | solved |
| 12/14/19 | Generating Word Cloud | <https://amueller.github.io/word_cloud/auto_examples/simple.html> | solved |
| 12/14/19 | Fixing Overlapping labels in matplotlib | https://stackoverflow.com/questions/26700598/matplotlib-showing-x-tick-labels-overlapping-despite-best-efforts | solved |
| 12/8/19 | Using geopy distance function | https://geopy.readthedocs.io/en/stable/#module-geopy.distance | solved |
| 12/14/19 | Using Mapquest Static Map | https://developer.mapquest.com/documentation/static-map-api/v5/getting-started/ | solved |
| 12/1/19 | Pluralizing words with inflect | <https://pypi.org/project/inflect/> | solved |