Chloe Lee, Zachary Landau, and Norah Smith

Barbara Ericson

SI201

15 December 2025

UT 201 Final Project Report: Taste Trackers

GitHub Repo Link: https://github.com/28zlandau/TasteTrackers_SI201FinalProject_ZCN

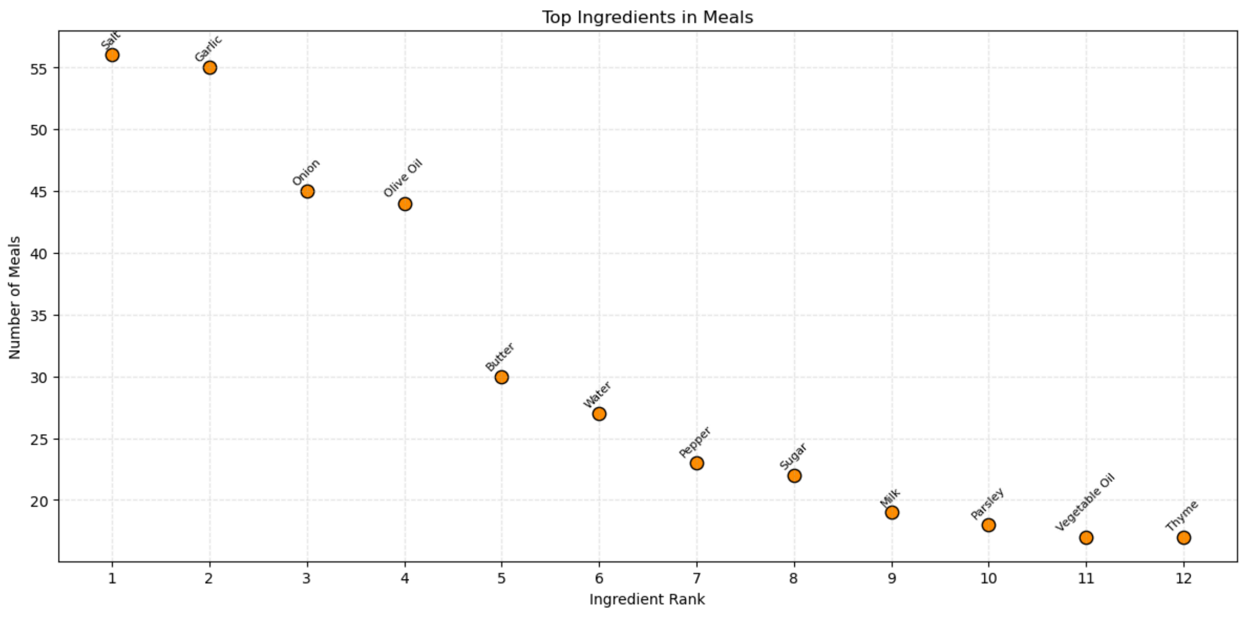
1. Project goals

Our goal for this project was to build a single database that combines data from 3 different public APIs and then calculate statistics across all tables to understand global patterns in meals, cocktails, and breweries.

Our API's and their uses:

- TheMealDB API - to gather meals, ingredients, categories, and areas
- TheCocktailDB API - to gather cocktails, categories, and glass types
- OpenBreweryDB API - to gather breweries, brewery types, and geographic location

We planned to gather at least 100 meals, 100 cocktails, and 100 breweries, stored across normalized tables with shared integer keys. Also, visualizations that compare categories, ingredient frequency, brewery types, and geographic patterns.



2. Goals achieved

All planned APIs were successfully used, and all data was fully gathered into one SQLite database:

- load_meals() gathered 150 unique meals and linked them with ingredients through the join table MealIngredients
- load_cocktails() gathered 150 cocktails and mapped them to glass types and categories
- load_breweries() gathered 150 breweries with normalized city, state, country, type, and name fields

All tables include integer foreign keys, no duplicate strings, and all data loads batches of ≤ 25 items per file run. The tables were created and stored using data.py in a single database taste_trackers.db.

3. **Problems we faced**

- API inconsistencies

Some APIs returned missing fields, so we fixed this using normalize_string() and default lookup IDs

- Avoiding duplicates

We enforced uniqueness using INSERT OR IGNORE for every lookup table, and this ensured that we would have no duplicate strings in the database

- API limits

Since we could only insert $\leq$ 25 new items per run, we used a RunState table to save the last offset for each API so the program can continue loading new items without duplicating data

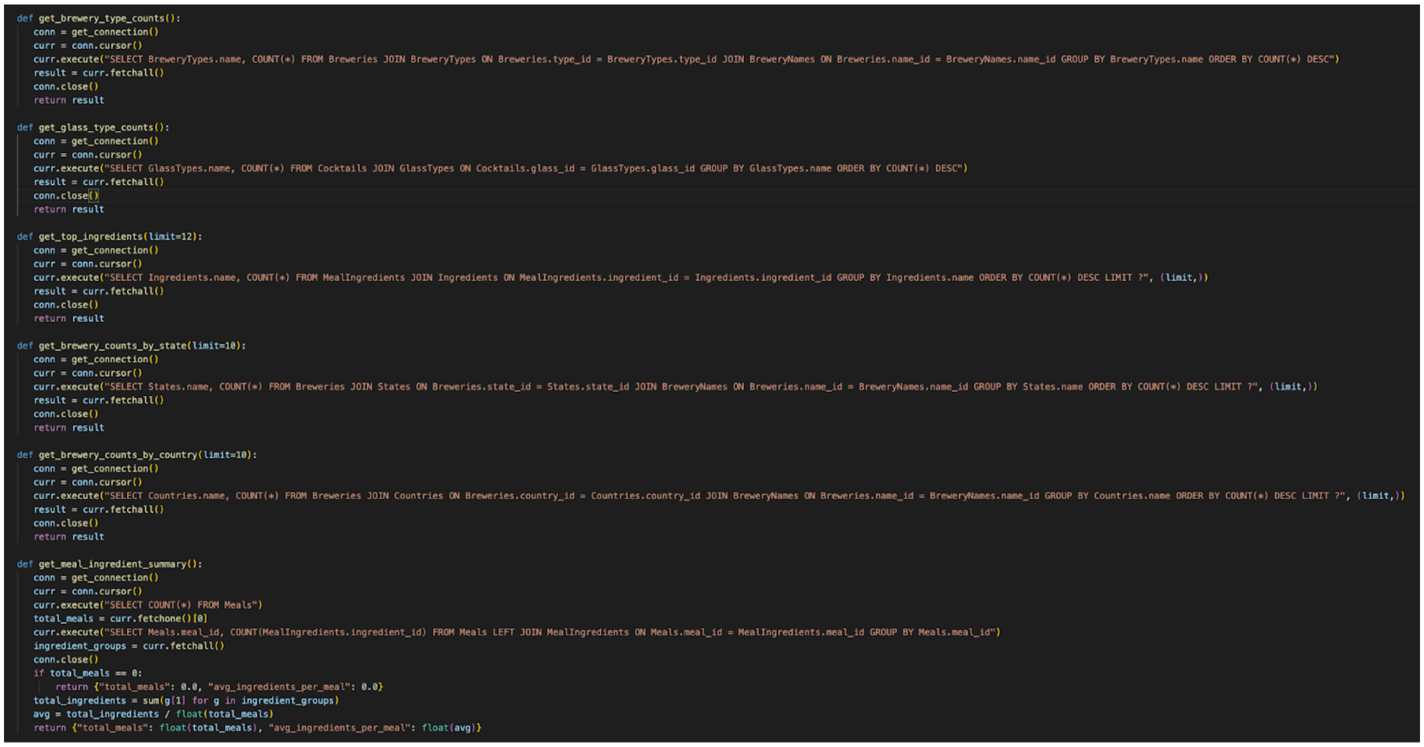
- Database joins

Some joins required enforcing consistent foreign keys, so we fixed this by using our lookup helper functions

4. **Calculated results**

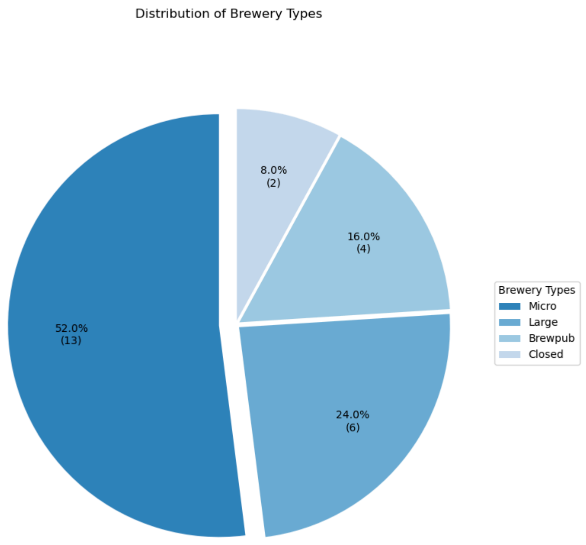
These results were generated automatically by write_calculations_to_file().

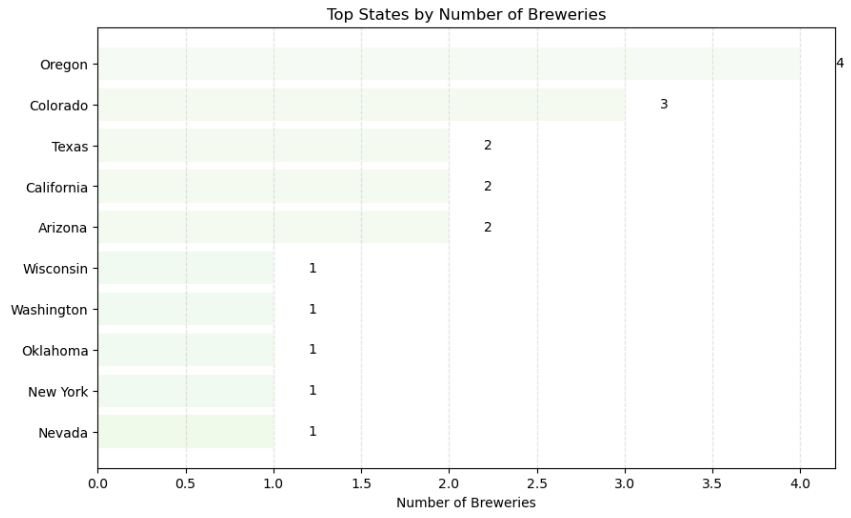
- Most common brewery type: Micro (13 breweries)
- Most common cocktail glass: Cocktail glass (12 cocktails)
- Top ingredient in meals: Salt (8 meals)
- Average ingredients per meal: 8.36
- Top state for breweries: Oregon (4)
- Top country: United States (23)

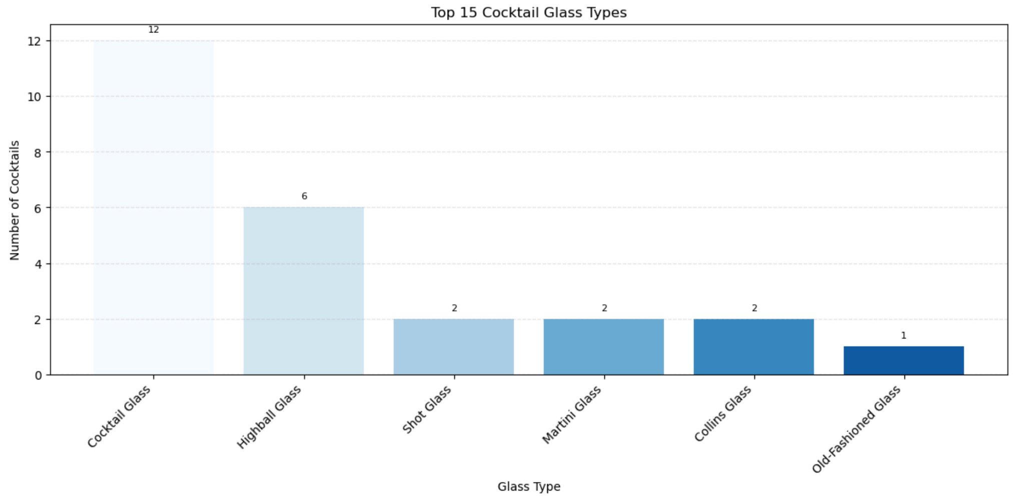


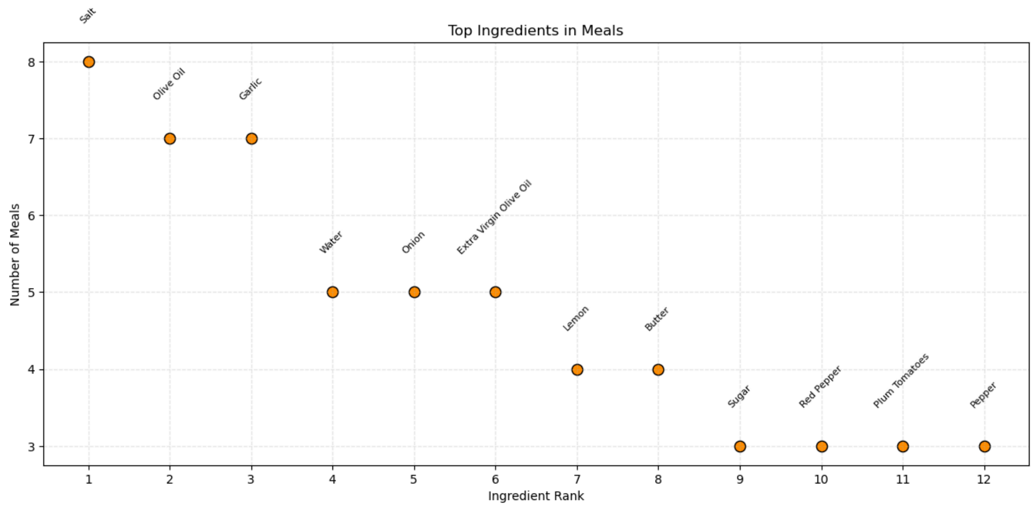
```python
def get_brewery_type_counts():
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT BreweryTypes.name, COUNT(*) FROM Breweries JOIN BreweryTypes ON Breweries.type_id = BreweryTypes.type_id JOIN BreweryNames ON Breweries.name_id = BreweryNames.name_id GROUP BY BreweryTypes.name ORDER BY COUNT(*) DESC")
    result = curr.fetchall()
    conn.close()
    return result

def get_glass_type_counts():
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT GlassTypes.name, COUNT(*) FROM Cocktails JOIN GlassTypes ON Cocktails.glass_id = GlassTypes.glass_id GROUP BY GlassTypes.name ORDER BY COUNT(*) DESC")
    result = curr.fetchall()
    conn.close()
    return result

def get_top_ingredients(limit=12):
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT Ingredients.name, COUNT(*) FROM MealIngredients JOIN Ingredients ON MealIngredients.ingredient_id = Ingredients.ingredient_id GROUP BY Ingredients.name ORDER BY COUNT(*) DESC LIMIT ?", (limit,))
    result = curr.fetchall()
    conn.close()
    return result

def get_brewery_counts_by_state(limit=10):
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT States.name, COUNT(*) FROM Breweries JOIN States ON Breweries.state_id = States.state_id JOIN BreweryNames ON Breweries.name_id = BreweryNames.name_id GROUP BY States.name ORDER BY COUNT(*) DESC LIMIT ?", (limit,))
    result = curr.fetchall()
    conn.close()
    return result

def get_brewery_counts_by_country(limit=10):
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT Countries.name, COUNT(*) FROM Breweries JOIN Countries ON Breweries.country_id = Countries.country_id JOIN BreweryNames ON Breweries.name_id = BreweryNames.name_id GROUP BY Countries.name ORDER BY COUNT(*) DESC LIMIT ?", (limit,))
    result = curr.fetchall()
    conn.close()
    return result

def get_meal_ingredient_summary():
    conn = get_connection()
    curr = conn.cursor()
    curr.execute("SELECT COUNT(*) FROM Meals")
    total_meals = curr.fetchone()[0]
    curr.execute("SELECT Meals.meal_id, COUNT(MealIngredients.ingredient_id) FROM Meals LEFT JOIN MealIngredients ON Meals.meal_id = MealIngredients.meal_id GROUP BY Meals.meal_id")
    ingredient_groups = curr.fetchall()
    conn.close()
    if total_meals == 0:
        return {"total_meals": 0.0, "avg_ingredients_per_meal": 0.0}
    total_ingredients = sum(g[1] for g in ingredient_groups)
    avg = total_ingredients / float(total_meals)
    return {"total_meals": float(total_meals), "avg_ingredients_per_meal": float(avg)}
```

5. Visualizations



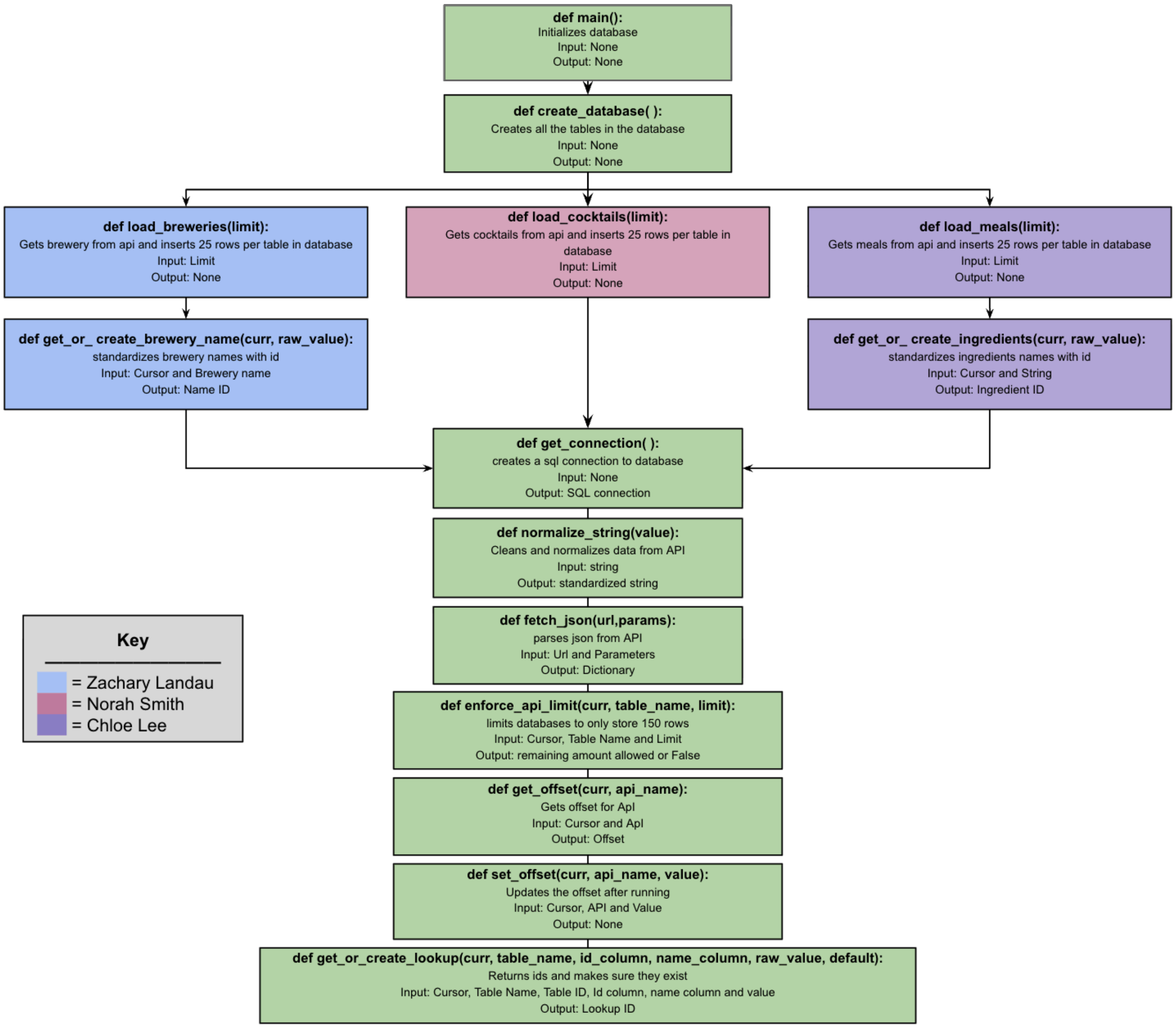

Top States by Number of Breweries
Oregon 4
Colorado 3
Texas 2
California 2
Arizona 2
Wisconsin 1
Washington 1
Oklahoma 1
New York 1
Nevada 1
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Number of Breweries


Top 15 Cocktail Glass Types
Number of Cocktails
0
2
4
6
8
10
12
Cocktail Glass 12
Highball Glass 6
Shot Glass 2
Martini Glass 2
Collins Glass 2
Old-Fashioned Glass 1
Glass Type

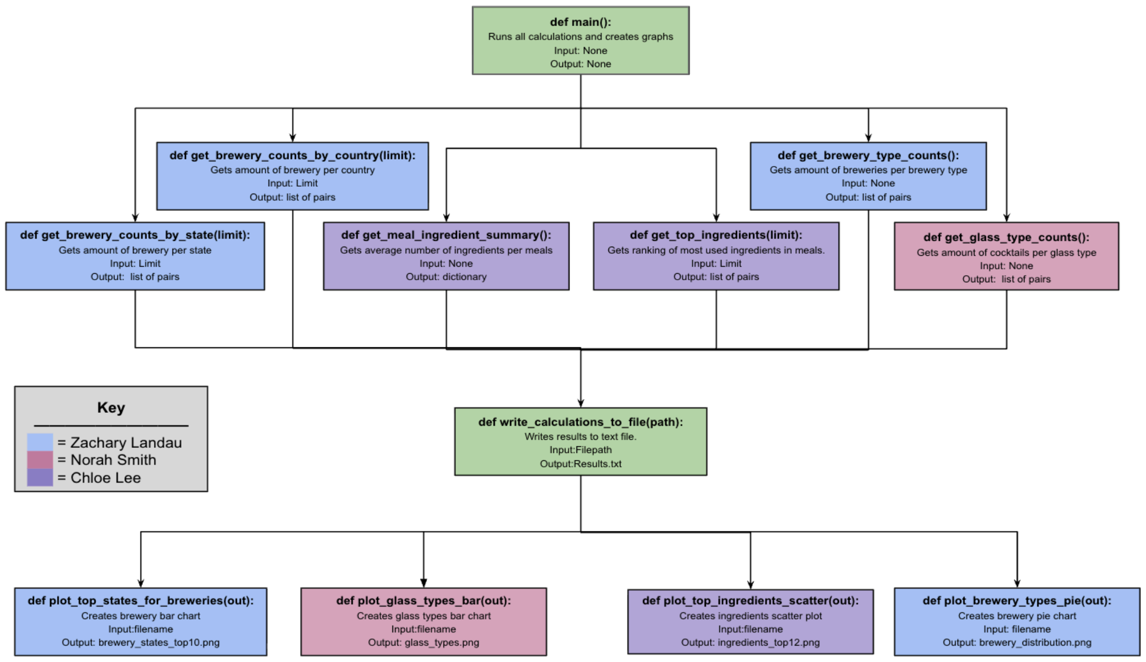

Top Ingredients in Meals
Number of Meals
3
4
5
6
7
8
Salt
Olive Oil
Garlic
Water
Onion
Extra Virgin Olive Oil
Lemon
Butter
Sugar
Red Pepper
Plum Tomatoes
Pepper
1
2
3
4
5
6
7
8
9
10
11
12
Ingredient Rank

6. How to run the code

To run this code, run the file data.py 6x (150 rows of data used being uploaded in 25 row increments) or until you see the message “All data uploaded to DB” in the VSCODE output. Next, run the file calculations_&_visuals.py and wait until it is done running. The visualizations and results summary will then appear.


def main():
Initializes database
Input: None
Output: None
def create_database( ):
Creates all the tables in the database
Input: None
Output: None
def load_breweries(limit):
Gets brewery from api and inserts 25 rows per table in database
Input: Limit
Output: None
def load_cocktails(limit):
Gets cocktails from api and inserts 25 rows per table in database
Input: Limit
Output: None
def load_meals(limit):
Gets meals from api and inserts 25 rows per table in database
Input: Limit
Output: None
def get_or_ create_brewery_name(curr, raw_value):
standardizes brewery names with id
Input: Cursor and Brewery name
Output: Name ID
def get_or_ create_ingredients(curr, raw_value):
standardizes ingredients names with id
Input: Cursor and String
Output: Ingredient ID
def get_connection( ):
creates a sql connection to database
Input: None
Output: SQL connection
def normalize_string(value):
Cleans and normalizes data from API
Input: string
Output: standardized string
def fetch_json(url,params):
parses json from API
Input: Url and Parameters
Output: Dictionary
def enforce_api_limit(curr, table_name, limit):
limits databases to only store 150 rows
Input: Cursor, Table Name and Limit
Output: remaining amount allowed or False
def get_offset(curr, api_name):
Gets offset for Apl
Input: Cursor and Apl
Output: Offset
def set_offset(curr, api_name, value):
Updates the offset after running
Input: Cursor, API and Value
Output: None
def get_or_create_lookup(curr, table_name, id_column, name_column, raw_value, default):
Returns ids and makes sure they exist
Input: Cursor, Table Name, Table ID, Id column, name column and value
Output: Lookup ID
Key
= Zachary Landau
= Norah Smith
= Chloe Lee

7. Updated function diagram


def main():
Runs all calculations and creates graphs
Input: None
Output: None
def get_brewery_counts_by_country(limit):
Gets amount of brewery per country
Input: Limit
Output: list of pairs
def get_brewery_type_counts():
Gets amount of breweries per brewery type
Input: None
Output: list of pairs
def get_brewery_counts_by_state(limit):
Gets amount of brewery per state
Input: Limit
Output: list of pairs
def get_meal_ingredient_summary():
Gets average number of ingredients per meals
Input: None
Output: dictionary
def get_top_ingredients(limit):
Gets ranking of most used ingredients in meals.
Input: Limit
Output: list of pairs
def get_glass_type_counts():
Gets amount of cocktails per glass type
Input: None
Output: list of pairs
Key
= Zachary Landau
= Norah Smith
= Chloe Lee
def write_calculations_to_file(path):
Writes results to text file.
Input:Filepath
Output:Results.txt
def plot_top_states_for_breweries(out):
Creates brewery bar chart
Input:filename
Output: brewery_states_top10.png
def plot_glass_types_bar(out):
Creates glass types bar chart
Input:filename
Output: glass_types.png
def plot_top_ingredients_scatter(out):
Creates ingredients scatter plot
Input:filename
Output: ingredients_top12.png
def plot_brewery_types_pie(out):
Creates brewery pie chart
Input: filename
Output: brewery_distribution.png

8. Documentation of external resources used

Date	Issue description	Location of resource	Result
11/20	I wasn't sure how to load only a small amount of data at a time from each API	API documentation pages for TheMealDB, TheCocktailDB, and OpenBreweryDB	Yes, it helped us understand how to request data in smaller groups
11/22	Ingredients and other fields had duplicates or empty text, which caused problems in the database	StackOverflow post about using INSERT OR IGNORE and UNIQUE in SQLite	Yes, it fixed the duplicate string problems
11/23	Some pages from the Brewery API weren't loading in order, and I didn't know how to move to the next page	OpenBreweryDB documentation about paging through results	Yes, it helped us load more breweries without repeating data
11/25	Matplotlib graphs needed better labels and colors	Matplotlib documentation (examples for bar charts, scatter plots, and pie charts)	Yes, it helped us format the charts so they look cleaner and more visually stimulating