Recipe101 - Recipe Recommendation System

Team Members:

- Le Pan: **lepan@seas.upenn.edu**
- Wenya Xiao: **wenyax@seas.upenn.edu**
- Songyu Yan: **yveyan@seas.upenn.edu**
- Lihan Zhu: **zhulihan@seas.upenn.edu**

---

1 Introduction

1.1 Project Motivation

Staring at the jammed fridge filled with ingredients seemingly can sustain a year-long supply, we have learned, in this post pandemic era, to shop as much as we can but never been taught how to use them up. We have drained up our creativity pondering - What should I eat today.

To help salvage poor users from the choice of food perplexity, we proposed a website that performs ingredient-based food recommendation.

1.2 Features

Guided by our motivation, we will implemented following features within our project:

1. 'What to cook?' - Two-step recipe recommendation based on ingredients and cooking time.
   This question is the core engine of our website. Our website allows users to search for recipes based on a list of ingredients. We are hoping to automate a ârecipe rankâ algorithm that generates a synthesized decision based on: user inputted ingredients, recipe ratings and food preparation time.
2. 'Cooking novice or master?' - recommend recipes based on the level of cooking difficulty.
   Even though levels of cooking difficulty are not explicitly listed within given data, we could implicitly infer this attribute based on cooking time, number of steps taken and the variety of ingredients. This features will be implemented by our most complex query.
3. 'Fan for a writer' - find recipes written by the same writer.
   After user selecting a specific recipe, our platform will prompt a series of other recipes written by the same author. The recipes will be listed by their average rating in descending order.
4. 'Star writer' - list recipes based on their writers' rating.
   As we expect users to maintain a higher level of stickiness to certain groups of recipe writers, we would like to implement a recipe recommendation mode based on recipe writer, list recipes written by the same writer sorted by their ratings.
5. 'Recipe you might also like' - similarity-based recipe recommendation.
   After performing an ingredient-based search, our website will also generate ârecipes you might likeâ. This is achieved finding recipes with at least 90% of overlapping ingredients.

2 Data

2.1 Data Sources

Our project involves data from 3 sources:

2.1.1 Recipe Ingredients - Kaggle: Recipe Ingredients and Reviews recipes.csv

This dataset that contains 12201 distinct recipes, 11 features including recipes name, review count, recipe photo, author, prepare time, cook time, total time, ingredients, and directions.

2.1.2 Recipe Reviews - Kaggle: Recipe Ingredients and Reviews reviews.csv

A dataset that contains 1616884 unique recipes, 4 features including recipes Id, consumer profile ID, their comment and ratings.

2.1.3 Spoonacular API - Spoonacular Food API

We used Spoonacular food API as a supplementary dataset for showing images for ingredients. After sending the name of an ingredient as a string, this API returns a URL of the food image.

2.2 Data Preparation

The **Recipe** and **Review** tables scripted from Kaggle contain missing values, mixed data types and text information unsuitable for Relational Database querying. Cleaning the data before importing to SQL is necessary.

2.2.1 Recipe Ingredients

We took following steps to preprocess this table:

1. Validate if RecipeID is unique, Review Count is positive.
2. Drop records without cooking directions, replacing missing time features with N/A.
3. Turn time *Prepare Time*, *Cook Time*, *Total Time* from strings to float measured in minutes.
4. Processing Ingredients is the most challenging task in the data preparation stage. In our raw dataset, ingredients are listed in the format of {quantity} {measurement unit} {ingredient name} {optional: processing steps (thawed/whipped)}.

   ```
   recipes_df.iloc[0]['Ingredients'] =
       '2 (.25 ounce) packages active dry yeast**3/4 cup warm water (110 degrees F/45 degrees
   C)**1/2 cup white sugar**1 teaspoon salt**2 eggs**1/2 cup butter, room temperature**4
   cups all-purpose flour**1/4 cup butter, softened'\\
   ```

   As we will be implementing recipe search using ingredients, we need to extract the ingredient name independently. We will be performing this task using **regular expression** match and string manipulations. We vision our cleaned dataset will have an independent ingredient sub-table with column recipeID, quantity, unit, ingredient.

RecipeID	quantity	Unit	Ingredient
7000	0.25	ounce	yeast
7000	1.5	cup	water
7000	1.0	cup	sugar
7000	1.0	teaspoon	salt

Figure 1: Ingredient per Recipe after cleaning

5. To meet BCNF design approach, we dropped ReviewCount, Ingredients from recipe dataframe, and created a new ingredient_recipe table to store the 'has' relationship between recipes and ingredients. After pre-processing the data, we arrived at 2 tables of following schema:

Table 1: Resulting Review Data Frame After Data Pre-processing

ColumnName	Interpretation	DataType
RecipeID	Uniquely identify a recipe	Integer
RecipeName	Name of Recipes	String
RecipePhoto	URL link to recipe photo	String
Author	Name of recipe writer	String
Directions	**separated string	String
PrepareTime	Total preparation time in minutes	Decimal
CookTime	Total cooking time in minutes	Decimal
TotalTime	Total time in minutes	Decimal

Table 2: Resulting Ingredient Data Frame After Data Pre-processing

ColumnName	Interpretation	DataType
RecipeID	References RecipeID in recipe table	Integer
Quantity	Ingredient quantity in unit	Decimal
Unit	Food measurement unit	String
Ingredient	Ingredient Name	String

2.2.2 Recipe Review

We took following steps to preprocess this table:

1. Validating if all RecipeID in Review table exists in Recipe table, remove unmatched reviews.
2. ProfileID, Rate, Comment columns contain missing values. As our main interest with Review is recipe ratings, we will drop rows without Rate. After dropping, we noticed that all missing values from Review have been removed.
3. We will drop Comment as we do not plan to implement this feature in our website

3 Database - Relational Schema and Entity Resolution

We ended up with a relational schema:

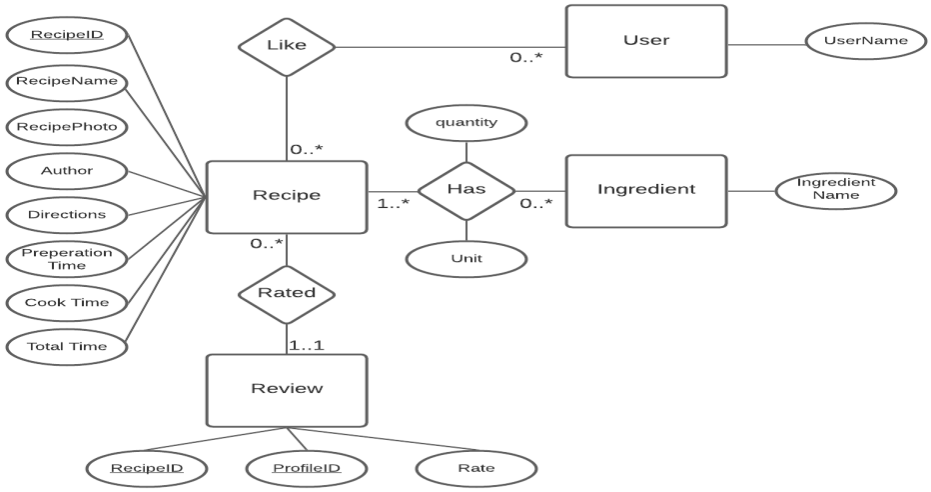

Figure 2: ER Diagram

recipes_cleaned (RecipeID, Recipe Name, Recipe Photo, Author, Directions, Preperation Time, Cook Time, Total Time)
reviews_cleaned(RecipeID, ProfileID, Rate, RecipeID REFERENCES recipes _cleaned(RecipeID))
ingredient_recipe(RecipeID, quantity, unit, ingredient, RecipeID REFERENCES recipes _cleaned(RecipeID))
user_favorite(userName, recipeID, recipeID RERERENCES recipes_cleaned(RecipeID))

In addition to tables created after importing given data, we also have a **user_favorite** table that allows logged-in users to dynamically add recipes to their favorite recipe list.

The **recipes_cleaned** contains 17607 distinct recipes; **reviews_cleaned** contains 1048575 reviews; **ingredient_recipe** contains 146604 records mapping ingredients to recipes. In our data pre-processing stage, we performed validity checking to assure that all *RecipeID* in **reviews_cleaned** exists within **recipes_cleaned**, and all *RecipeID* within **ingredient_recipe** exists within **recipes_cleaned**. We also checked no repetitive ratings were given by the same ProfileID on one RecipeID.

Our relational schema conforms to BCNF normalization. Each recipe in **recipes_cleaned** depends solely on *RecipeID*. Each review under **reviews_cleaned** depends solely on *RecipeID* and *ProfileID*. **ingredient_recipe** represents a 'has' relationship between ingredients and recipes, each relation is uniquely identified by *RecipeID*. **user_favorite** stores list of 'liked' recipes from a logged-in user, uniquely identified by *userName*. As all relations depends solely on the key or a superkey, this schema meets the requirement of BCNF.

4 Queries

Our queries are designed to realize 5 features we proposed within the **Feature**1.2 section.

1. 'What to cook?' - Two-step recipe recommendation based on ingredient and cooking time. After user entering the name of 1 ingredient, we use string pattern matching and group concatenation to find recipes containing given ingredient.

```
WITH ingred_name AS(
    SELECT recipeID, GROUP_CONCAT(ingredient SEPARATOR ',') ings
    FROM ingredient_recipe
    GROUP BY recipeID
)
SELECT IR.RecipeID, `Recipe Name`, `Recipe Photo`, Author,  Directions, Total_Time
FROM recipes_cleaned RC JOIN ingred_name IR ON RC.recipeID=IR.recipeID
WHERE ings LIKE '%$user_input_ingredient$%'
LIMIT 5;
```

We also allow user to select recipes based on total cooking time.

```
SELECT RecipeID, Recipe Name, Recipe Photo, Author, Directions, Total_Time
FROM recipes_cleaned
WHERE Total_Time BETWEEN $user_enter_lower_time$ AND $user_enter_upper_time$
LIMIT 20;
```

2. 'Cooking novice or master?' - recommend recipes based on level of cooking difficulty. This is our **most complex** query within this project. It aims to find recipes based on a user chosen level of cooking difficulty. Difficulty is defined by total cooking time and number of ingredients used. To recommend recipes with higher quality, we firstly selected a group of 'picky users', those who have a total average rating below population average rating. From these 'picky users', we found recipes rated above their personal rating average. Finally, out of these selected recipes, we automated recipe recommendation based on difficulty.

```
WITH avg_rating AS (
    SELECT avg(rate) FROM reviews_cleaned
), negative_user AS(
    SELECT profileID, avg(rate) as user_avg_rate
    FROM (SELECT profileID, rate FROM reviews_cleaned) AS RC
    GROUP BY profileID
    HAVING avg(rate)<=(SELECT * FROM avg_rating)
), recipe_ratings AS(
    SELECT RecipeID, avg(Rate) as avg_rate
    FROM reviews_cleaned RC RIGHT JOIN negative_user NU ON
        RC.profileID=NU.profileID AND RC.rate>NU.user_avg_rate
    GROUP BY RecipeID
), recipe_difficulty AS(
    SELECT RR.RecipeID,
        CASE WHEN Total_Time<30 AND COUNT(IR.ingredient_id)<10 THEN 'easy'
        WHEN Total_Time<30 AND COUNT(IR.ingredient_id)<20 THEN 'medium'
        ELSE 'hard' END as difficulty,
        RR.avg_rate AS rating
    FROM recipe_ratings RR
        JOIN recipes_cleaned RC ON RR.recipeID=RC.recipeID
        RIGHT JOIN ingredient_recipe IR ON RR.recipeID=IR.recipeID
    GROUP BY RR.RecipeID, RC.Total_Time, RR.avg_rate
)
SELECT *
    FROM recipe_difficulty RD LEFT JOIN recipes_cleaned RC ON RD.recipeID=RC.recipeID
    WHERE difficulty= $user_select_level_of_difficulty$
    ORDER BY rating DESC LIMIT 10;
```

3. 'Fan for a writer' - find recipes written by the same writer. After user lands on the page of a specific recipe, the platflom generates other recipes from the same writer and can be completed within similar cooking time (t±80).

```
SELECT RecipeID, Recipe Name, Recipe Photo, Author, Directions, Total_Time
```

```
FROM recipes_cleaned
WHERE Author = $user_select_recipe_author$ AND Total_Time
    BETWEEN $user_select_Time$-80 AND $user_select_Time$+80
LIMIT 20;
```

4. 'Star writer' - find recipes written by the same writer. This query recommends highest rated authors to users.

```
SELECT a.Author, avg(b.Rate)
FROM recipes_cleaned a JOIN reviews_cleaned b
ON a.RecipeID = b.RecipeID
GROUP BY a.Author
ORDER BY avg(b.Rate) DESC
LIMIT 20 ;
```

5. 'Recipe you might also like' - similarity-based recipe recommendation. After user lands on the page for a specific recipe, we find a series of other recipes with 90% ingredients overlapping. We return a list of similar recipes based on their averaged ratings.

```
WITH ingredients AS(
    SELECT ingredient
    FROM ingredient_recipe
    WHERE RecipeID=$user_enter_recipeID$
), similar_ingredient_num AS(
    SELECT FLOOR(COUNT(*)*0.9)
    FROM ingredients
), similar_recipe AS(
    SELECT RecipeID
FROM ingredient_recipe
GROUP BY recipeID
HAVING
    COUNT(ingredient in (SELECT * FROM ingredients))>=
    (SELECT *
    FROM similar_ingredient_num)
), reviews_avg AS(
    SELECT RecipeID, avg(rate) as avg_rate
    FROM reviews_cleaned
    GROUP BY RecipeID
)
SELECT DISTINCT SR.recipeID as ID, SC.‘Recipe Name‘ as RecipeName,
    SC.‘Recipe Photo‘ as RecipePhoto
FROM similar_recipe SR
 LEFT JOIN recipes_cleaned SC ON SR.recipeID=SC.recipeID
 LEFT JOIN reviews_avg RC ON SR.recipeID=RC.recipeID
WHERE SR.recipeID!=$user_enter_recipeID$
GROUP BY SR.recipeID, SC.‘Recipe Name‘, SC.‘Recipe Photo‘, RC.avg_rate
ORDER BY avg_rate DESC;
```

5 Performance Optimization

We focused on optimizing query[2] and query[5]. We significantly decreased the query fetching time by on average 60% using multiple query optimization methods.

1. We re-ordered join operations so that cross joins are avoided and smaller relations are always placed as the outer relation.
2. We pushed down projections, shrinking size of relation before join operations.
3. We cached temporary results by using multiple common table expressions, and reduced number of features by projecting only on critical columns.

4. If possible, we limited records returned from common table expressions and delayed sorting till the last selection.

Table 3: query[2] performance comparison

Test Input	Before	After
'easy'	8.062	3.26
'medium'	11.062	3.25
'hard'	7.313	3.5

Table 4: query[5] performance comparison

Test Input	Before	After
7200	2.765	1.203
7300	2.719	1.188
7500	2.391	1.234

6 Architecture

6.1 Technologies

1. For data storage, we used MySQL and AWS RDS. For AWS RDS setup process, we selected MySQL mode and created master username and password, then chose allocated storage to be 20 GB. We connected MySQL to the AWS database, and populated data from local .csv files.
2. For front-end implementation, we used node.js and React.js.

6.2 Website Structure

6.2.1 Login Page

With the URL (/login), users can access the login page. This page is mainly responsible for the login function. Username and password are required to be entered for the login function. After a successful login, the web page will be rerouted to the homepage, URL(/home). A link to the registration page (/register) is provided for new users. Credentials will be stored in the local storage after login and will not be deleted until the user chooses to log out. The user will be able to access the homepage as long as his credential is saved in the cookies.

Furthermore, the credential is used in the function for fetching and storing usersâ favorite recipes from and to the database.

6.2.2 Registration Page

With the URL(/register), users can access the registration page. This page is mainly responsible for the registration. Username, password, and password confirmation are needed to be entered for registration. The password is required to have at least one character, one number, and a length greater than 5. Satisfying all the requirements, users will be allowed to click the Register button. If there is no already existed account, the user will successfully create an account and be re rerouted to the login page (/login). A link to the login page (/login) is provided for users who already have accounts.

6.2.3 Navigation bar

A navigation bar is provided on the top of all pages except the login and registration page to provide free access to all functional components of the web. Five tags listed in the navigation bar are âHomeâ, âSearchâ, âBestâ, âCollectionâ, and âLogoutâ, corresponding to the web URL â/homeâ, â/searchâ, â/bestâ, âcollectionâ and a button for log out.

6.2.4 Homepage

With the URL(/home), the user can access the home page. The homepage serves the same purpose as the navigation bar, but in addition, it gives the introduction of our web function in detail. Users can click aˆlearn moreˆa to access the corresponding functional components of the web.

Figure 3: Homepage

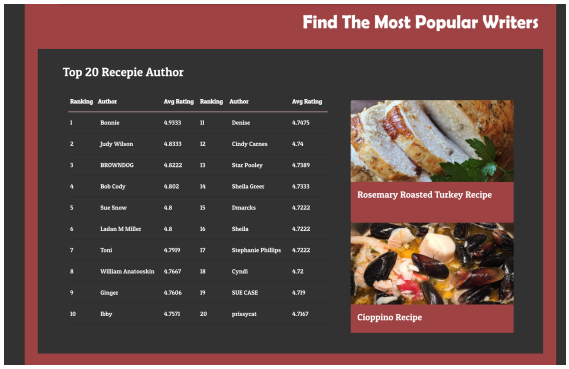

Figure 4: Best Author Page

6.2.5 Search page

With the URL(/search), users can access the search page. The search page provides three options for users to search for recipes. For the first option, users can enter up to four ingredients and search recipes based on these ingredients. For each ingredient entered, the website will provide 5 recipes. For the second option, users are given three choices, easy, medium, or hard to do recipe search based on cooking level. For the last option, users can choose a cooking time between zero to a time greater than 3.5 hours and search recipes base on the time. All recipes returned will be listed with a name, a keyword, a total time cost, and a button to learn more. By clicking the button aˆLEARN MOREˆa, users will be rerouted to a single recipe page to get more information about the chosen recipe and some of the webˆas recommendations. Single Recipe Page:

With the URL format in (/learnmore/?id=author= totaltime=), user can access the single recipe page of a specific recipe. On this page, all detailed information about the recipe will be provided including a name, an id, an author, a table of ingredients, and a list of cooking directions. Two of the websiteâs main recommendation functions are embedded in this page. Recommended recipes based on the same author and similar cooking times are listed. A list of recommended recipes based on 90% matching ingredients is provided. Users can choose to favorite or dislike the recipe. Favoriting a recipe will create a relation between the recipe and the user, and it will be stored in the database. Disliking a recipe will result in deleting the relation of the two.

6.2.6 Best Author Page

With the URL (/best), users can access the best author page of the web. On the mounding of the page, a list of top 20 recipe authors with ranking, author, and average rating will be provided. Users can further click an author name to display his recipe lists on the right of the page. By clicking on the recipe in the list, the user can learn more about the recipe on the single recipe page.

6.2.7 Collection Page

With the URL(/collection), users can access the collection page. All userâs favorite recipes saved in the database will be fetched and listed on the page. Users can click each recipe for accessing the single recipe page for more information. If there is no favorite recipe, the page will display a message âYou currently don't have any Favorite Item, Go and Exploreâ.

7 Technical Challenges

1. Preparing data for SQL importation was challenging. The raw data collected from Kaggle contains a large number of missing values, mixed variable type and none-atomic values. We used Pandas, nltk, regex in Python to transform data to desirable format. Please refer to our .pynb notebook for detailed data cleansing methods.
2. As we intend to enable the website to display images for all ingredients, with 3140 distinct ingredients, storing these images locally is expensive. We instead opted to use the Spoonacular API, which provides url to food images.
3. Designing complex SQL queries was not easy for us. As our project is search-based rather than anlytical, most features can be implemented via relatively less sophisticated SQL queries. We created our complex SQL queries by asking more creative business questions and adding more constrains to the search result.

8 Extra Credit

1. We integrated the Spoonacular API into our website for showing food images.
2. We implemented user-login functionality that allows user to add or remove recipes to their own favorite list.

9 Acknowledgement

We appreciate the assistance and guidance from Prof. Susan Davidson and every teaching assistant throughout the semester. Thank you for making this semester a pleasant and rewarding journey.