CCT College Dublin

Assessment Cover Page

Module Title:	Data Visualization Techniques, Machine Learning for Business
Assessment Title:	CA2
Lecturer Name:	David McQuaid, Muhammad Iqbal
Student Full Name:	Arthur Claudino Gomes de Assis
Student Number:	2023146
Assessment Due Date:	01/12/2023
Date of Submission:	05/12/2023

Declaration

TOTAL WORDS ON REPORT: 2144

1 INTRODUCTION AND BUSINESS UNDERSTANDING

Retailing is the selling of goods and services to consumer end users, according to (Montevirgen, 2023). Nowadays, the acquisition of data from users permits companies to achieve a certain level of knowledge about people navigating through internet that has made possible to them to create a customized experience to each person, turning them in potential customers. This experience comes often through advertising, customized according to each person personal data collected from diverse sources, including navigating history, previous online purchase activity, and even voice data collected from our mobile devices.

To use this data in an efficient way, retailers use Machine Learning models to understand peoples preferences, and recommend products according to behaviour and to their personal data. These Machine Learning models are called Recommendation Systems.

The dataset that will be explored in this project can be found in the link https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions/data?select=interactions_validation.csv, and it refers to data collected from a website that provides culinary recipes to users, in a format of a blog. The dataset was chosen because it contains similar features to retailers. Although it does not aim to sell its recipes, blogs are one of the sources targeted by companies to expose advertising. According to (Mimi Polner, 2023), blogs can be used to bring in sales or an income, and bloggers bring in an average annual income of $37,073. Some of the ways explored by the article that brings income to bloggers are Advertising Networks, Digital Products, Affiliate Links or Codes and Premium Content. We can therefore, consider blogs as a direct advertising retail business, in which the blogger aim to sell space for advertising in its website, and the quality of the content and experience will bring in more viewers, potentially increasing profits.

When we first verified our dataset, we noticed it contains 231,637 recipes! That may generate confusion on users that might head towards a more classified source of information. This is when recommendation systems come to action.

```
#Importing both dataset that will be used on the analysis
df_recipes = pd.read_csv('/Users/arthurassis/Documents/CCT - Data Analytics for Business/Machine Learning/Semest
df_users = pd.read_csv('/Users/arthurassis/Documents/CCT - Data Analytics for Business/Machine Learning/Semester
```

Words counting: 321.

2 Data Cleaning

```
#Verifying basic informations of the recipes dataset
df_recipes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231637 entries, 0 to 231636
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            231636 non-null  object
 1   id              231637 non-null  int64
 2   minutes         231637 non-null  int64
 3   contributor_id  231637 non-null  int64
 4   submitted       231637 non-null  object
 5   tags            231637 non-null  object
 6   nutrition       231637 non-null  object
 7   n_steps         231637 non-null  int64
 8   steps           231637 non-null  object
 9   description     226658 non-null  object
 10  ingredients     231637 non-null  object
 11  n_ingredients   231637 non-null  int64
dtypes: int64(5), object(7)
memory usage: 21.2+ MB
```

We can verify that our dataset contains 231,637 observations and 12 features. It seems that the data is nearly complete, except for 1 recipe that does not have name, and around 5,000 that does not have any description. We can drop those observations once they represent something around 2% of our dataset.

We also verify that our dataset has a date feature, that is read as an object, and it will be converted into datetime object to help in future analysis.

```
#Dropping null observations for the columns name and description.
df_recipes.name.dropna(inplace = True)
df_recipes.description.dropna(inplace = True)
```

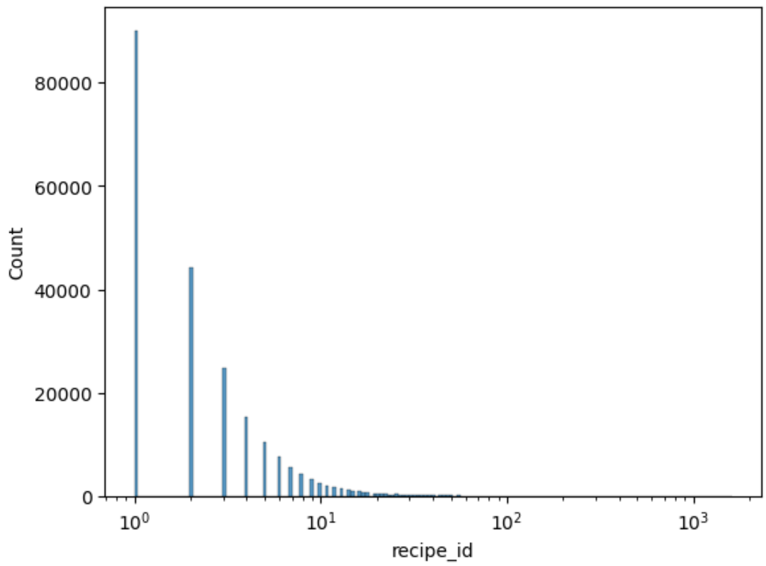
```
#Converting the column submitted into datetime object
df_recipes['submitted'] = pd.to_datetime(df_recipes['submitted'])
```

```
#Verifying basic informations of the interactions dataset
df_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1132367 entries, 0 to 1132366
Data columns (total 5 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   user_id    1132367 non-null  int64
 1   recipe_id  1132367 non-null  int64
 2   date       1132367 non-null  object
 3   rating     1132367 non-null  int64
 4   review     1132198 non-null  object
dtypes: int64(3), object(2)
memory usage: 43.2+ MB
```

It is possible to verify that in this case we have over a million observations and 5 features. Again the data is quite clean, and only less than 200 fields on reviews are null, what represents a negligible parcel of the whole, and there it will also be dropped. We will also convert the date column in this case to a datetime object.

We will now verify the number of times each recipe was reviewed.



We could verify that among our whole dataset, that contains over 200,000 recipes, only around 1/5 contain more than 5 reviews. It will be considered relevant in this dataset only the 40,000 most reviewed recipes. The main reason is that while performing the methods required on the scope of the project for recommendation systems, the computer used could not perform the operations with more than 40,000 observations of recipes.

```
#Filtering the most relevant 40000 reviews only on the dataset containing information of recipes
most_relevant_reviews = 40000
relevant_recipes = df_recipes.nlargest(most_relevant_reviews, 'n_reviews').reset_index()
```

We understand that the number of reviews refers to engagement in one specific page of our website. Therefore, using the information that we have on our dataset, if we are able to discover what increases the number of reviews, we can optimize our website to increase its views, and therefore sales.

Words counting : 276

3 Building a content based recommendation system

A content based recommendation is a system based exclusively in products characteristics. It bases its recommendations in the similarity existent between two products.

In our case, we will build a recommendation engine based in two features of our dataset: tags, and ingredients.

The first step is parsing our string to get a list of values. We can using the function literal_eval that reads the line as though it was a code, and extracts the meaning of it.

We then parse our data once more removing inconvenient information that may lead our model to bad results. (Leitch, 2020) discussed the issue of the ingredients for a recommendation systems of recipes utilizing a different dataset. He discusses that some words on our ingredients do not contribute to our recommendations, because they are very common in almost every recipe, such as oil, weights and measures. He proposes a parser for his own project that will be applied in our dataset. The parser was found on his Github.

```python
#Parser provided by (Leitch, 2020) on his Github.

# Weigths and measures are words that will not add value to the model. I got these standard words from
# https://en.wikibooks.org/wiki/Cookbook:Units_of_measurement

def ingredient_parser(ingreds):
    """

    This function takes in a list (but it is a string as it comes from pandas dataframe) of
       ingredients and performs some preprocessing.
       For example:

       input = '['1 x 1.6kg whole duck', '2 heaped teaspoons Chinese five-spice powder', '1 clementine',
                 '6 fresh bay leaves', 'GRAVY', '', '1 bulb of garlic', '2 carrots', '2 red onions',
                 '3 tablespoons plain flour', '100 ml Marsala', '1 litre organic chicken stock']'

       output = ['duck', 'chinese five spice powder', 'clementine', 'fresh bay leaf', 'gravy', 'garlic',
                 'carrot', 'red onion', 'plain flour', 'marsala', 'organic chicken stock']

    """
    measures = ['teaspoon', 't', 'tsp.', 'tablespoon', 'T'...]
    words_to_remove = ['fresh', 'oil', 'a', 'red', 'bunch'...]
    # The ingredient list is now a string so we need to turn it back into a list. We use ast.literal_eval
    if isinstance(ingreds, list):
        ingredients = ingreds
    else:
        ingredients = ast.literal_eval(ingreds)
```

```python
    # We first get rid of all the punctuation. We make use of str.maketrans. It takes three input
    # arguments 'x', 'y', 'z'. 'x' and 'y' must be equal-length strings and characters in 'x'
    # are replaced by characters in 'y'. 'z' is a string (string.punctuation here) where each character
    #  in the string is mapped to None.
    translator = str.maketrans("", "", string.punctuation)
    lemmatizer = WordNetLemmatizer()
    ingred_list = []
    for i in ingredients:
        i.translate(translator)
        # We split up with hyphens as well as spaces
        items = re.split(" |-", i)
        # Get rid of words containing non alphabet letters
        items = [word for word in items if word.isalpha()]
        # Turn everything to lowercase
        items = [word.lower() for word in items]
        # remove accents
        items = [
            unidecode.unidecode(word) for word in items
        ]  #''.join((c for c in unicodedata.normalize('NFD', items) if unicodedata.category(c) != 'Mn'))
        # Lemmatize words so we can compare words to measuring words
        items = [lemmatizer.lemmatize(word) for word in items]
        # Gets rid of measuring words/phrases, e.g. heaped teaspoon
        items = [word for word in items if word not in measures]
        # Get rid of common easy words
        items = [word for word in items if word not in words_to_remove]
        if items:
            ingred_list.append(" ".join(items))
    # ingred_list = " ".join(ingred_list)
    return ingred_list
```

```python
relevant_recipes['tags'][0]
```

```
['time-to-make',
 'course',
 'main-ingredient',
 'cuisine',
 'preparation',
 'north-american',
 'breads',
 'fruit',
 'american',
 'oven',
 'dietary',
 'quick-breads',
 'equipment',
 '4-hours-or-less']
```

```python
relevant_recipes['tags'][1000]
```

```
['30-minutes-or-less',
 'time-to-make',
 'course',
 'main-ingredient',
 'cuisine',
 'preparation',
 'occasion',
 'lunch',
 'side-dishes',
 'eggs-dairy',
 'rice',
 'easy',
 'european',
 'dinner-party',
```

```python
relevant_recipes['tags'][5000]
```

```
['60-minutes-or-less',
 'time-to-make',
 'course',
 'main-ingredient',
 'cuisine',
 'preparation',
 'occasion',
 'north-american',
 'low-protein',
 'healthy',
 'cobblers-and-crisps',
 'desserts',
 'fruit',
 'canadian',
```

We can verify through the examples above, that the first tags are usually very similar among themselves, what means that getting the first tags it is not the best idea, therefore, we will select 5 tags after the sixth one.

We can create now the metadata soup that will be feed in our vectorizer.

Our Vectorizer has found 2857 vocabularies used on the dataset according to the features we have used.

The next step is building a reverse mapping to get the name of our recommendations.

```python
# Function that takes in Recipe name as input and outputs most similar recipes
def get_recommendations(name, cosine_sim=cosine_sim):
    # Get the index of the recipe that matches the title
    idx = indices[name]

    # Get the pairwise similarity scores of all recipes with that recipe
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the recipes based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar recipes (0 is excluded because it would be the own recipe itself)
    sim_scores = sim_scores[1:11]

    # Get the recipes id
    recipes_id = [i[0] for i in sim_scores]

    # Return the top 10 most similar recipes
    return relevant_recipes['name'].iloc[recipes_id]
```

```python
get_recommendations('dark chocolate cake')
```

```
16028                       ora s deep dark chocolate cake
497                 hershey s chocolate cake with frosting
15265                                 chocolate snack cake
9781                          moms chocolate zucchini cake
12018                          the best chocolate snack cake
13231                       spago s chocolate chiffon cake
2349      ultra moist starbucks chocolate cake  or cupcakes
17350                 easy one bowl apple snack brownie cake
6520                          super moist chocolate spelt cake
216                                  mom s chocolate cupcakes
Name: name, dtype: object
```

As we can verify, we can find using our content recommender system recipes that are alike the one searched. Data for the searching engine could work as a link in a website. If one click on the recipe for dark chocolate cake, it will receive in some fields on the screen recommendations to related recipes.

Words counting: 305.

4 Creating a Collaborative Filtering

Collaborative Filtering, different of content recommendation systems, do not use products data to find similarity among products. Instead, it uses data from different users, and based on what users like or dislike, the recommendation is made. There are two mechanisms on Collaborative Filtering that can be used to create recommendations, the user-based and the item-based.

Both of them are based in the similarity of users previous actions, rather than the content of the product. For instance, in our business case, we may discover that users that have rated chicken based recipes, have also rated nuts based recipes. Once a new user access the system and rate by the first time a chicken recipe, our recommendation system will recommend nuts based recipe, rather for the similarity between users preferences than for the similarity between the recipes itself.

We will in this project build an user-based recommendation system.

In the first step, we want to filter back in our interactions dataset, only reviews that were written to recipes in our list of relevant recipes.

Verifying our new dataset, we can see it still contains over 700,000 observations and nearly 200,000 users! Due to computational costs, we will need to filter only the 7,500 top of our website.

```
#Grouping our dataset by users_id to filter only the 7,500 most common users.
users_count = df_validusers.groupby(by = 'user_id').size().reset_index()
```

```
#Renaming the columns count created
users_count.rename(columns = {0:'count'}, inplace = True)
```

```
#Defining our threshold
most_common_users = 7500
```

```
#Extracting the common users based on our threshold
common_users = users_count.nlargest(most_common_users, 'count').reset_index()
```

```
#Getting a list of the common users_id
common_users_id = common_users['user_id'].values.tolist()
```

```
#Filtering our valid users dataset into a sample dataset
df_sampleusers = df_validusers[df_validusers['user_id'].isin(common_users_id)]
```

Now that we reduced our dataset only to the top 7,500 users, we will apply the methods to seek for recommendations for a certain user, based on its similarity to other users.

The methods is based in a few steps:

1- Finding the average user in recipes they have reviewed.
2- Calculating an adjusted average for each user and recipe based in its average and its rating to the recipe.
3- Creating a matrix of users x recipes, and its adjusted average on the values.
4- Filling all recipes unreviewed by each user with an average of the adjusted average for the user.
5- Finding the similarity using this matrix through cosine similarity.
6- Sorting similarity for each user, and mapping index into recipes names.

After performing this process, we get the following results to a random user 353579.

```
Enter the user id to whom you want to recommend : 353579

The Recommendations for User Id : 353579

potato kielbasa skillet
real southern cornbread
cornbread dressing
broccoli salad with gouda
shepherd s pie oamc
```

Let’s see and compare some results we get when using the content based recommendation system for two of this recipes, and compare with another recipe provided by the recommendation.

```
df_recipes[df_recipes['name'] == 'fried potatoes and smoked sausage']
```

	name	recipe_id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients	rating	n_reviews
87830	fried potatoes and smoked sausage	374959	35	574635	2009-05-29	['60-minutes-or-less', 'time-to-make', 'course...	[631.5, 50.0, 18.0, 32.0, 37.0, 46.0, 22.0]	10	['put a large skillet on the stove on med high...	yummy comfort food, that the whole family will...	['potatoes', 'smoked sausage', 'onion', 'garli...	6	4.115385	26

```
print(df_recipes.iloc[87830,5])
```

```
['60-minutes-or-less', 'time-to-make', 'course', 'main-ingredient', 'cuisine', 'preparation', 'occasion', 'north-am
erican', 'main-dish', 'pork', 'potatoes', 'vegetables', 'easy', 'beginner-cook', 'kid-friendly', 'dietary', 'one-di
sh-meal', 'low-cholesterol', 'comfort-food', 'inexpensive', 'healthy-2', 'toddler-friendly', 'low-in-something', 'm
eat', 'pork-sausage', 'taste-mood', 'savory', '3-steps-or-less']
```

```
df_recipes[df_recipes['name'] == 'potato kielbasa skillet']
```

	name	recipe_id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients	rating	n_revie
164824	potato kielbasa skillet	34688	28	27643	2002-07-23	['30-minutes-or-less', 'time-to-make', 'course...	[789.3, 58.0, 32.0, 56.0, 47.0, 57.0, 29.0]	6	['in dutch oven , brown potatoes , kielbasa , ...	a very hearty, tasty supper. this is very easy...	['potatoes', 'kielbasa', 'onion', 'green peppe...	11	4.309524	

```
print(df_recipes.iloc[164824,5])
```

```
['30-minutes-or-less', 'time-to-make', 'course', 'main-ingredient', 'cuisine', 'preparation', 'occasion', 'north-am
erican', 'main-dish', 'eggs-dairy', 'pork', 'potatoes', 'vegetables', 'american', 'southern-united-states', 'easy',
'european', 'kid-friendly', 'polish', 'cheese', 'stove-top', 'dietary', 'comfort-food', 'inexpensive', 'meat', 'por
k-sausage', 'taste-mood', 'equipment']
```

```
df_recipes[df_recipes['name'] == 'real southern cornbread']
```

	name	recipe_id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients	rating	n_revi
171645	real southern cornbread	51550	35	38532	2003-01-16	['60-minutes-or-less', 'time-to-make', 'course...	[235.1, 17.0, 20.0, 12.0, 9.0, 16.0, 9.0]	15	['put the skillet in the oven and preheat to 3...	this is my families recipe, with a small chang...	['yellow cornmeal', 'unbleached all-purpose fl...	10	4.492647	

```
print(df_recipes.iloc[171645,5])
```

```
['60-minutes-or-less', 'time-to-make', 'course', 'cuisine', 'preparation', 'occasion', 'north-american', 'breads',
'american', 'southern-united-states', 'holiday-event', 'dietary', 'high-calcium', 'quick-breads', 'high-in-somethin
g', 'kwanzaa']
```

It is possible to verify in our example, that when we get recommendations based on content for potato kielbasa skillet, we get recipes that are similar to the first, taking the example of fried potatoes and smoked sausage, we can see they apparently have similar ingredients and they are tagged with similar tags. On the other hand, we do not find anything similar between the two first dishes recommended for the user 353579 as expected.

Words counting: 448.

5 Market Basket Analysis

It will be used in our case, the list of ingredients for recipes as our basket. It is interesting to analyse this scenario because companies that sell professional products for chefs or restaurants may benefit from the analysis verifying which kind of products are necessary together in order to produce certain food.

	Ingredients	count
2489	onion	13328
3623	sugar	12574
1441	garlic	12405
463	butter	11806
1216	egg	9730
1344	flour	8654
2257	milk	5540
3808	tomato	4394
1006	cream	4378
1985	lemon	4227
3964	vegetable	3066

We can verify that our most consumed ingredients are basic in any recipes, to season, and they are not products with a high value of market.

Using the algorithms that perform the Apriori principle in our dataset, we get the following results.

```
frequent_itemsets_ap.sort_values(by = 'support', ascending = False).head(20)
```

	support	itemsets
84	0.333200	(onion)
124	0.314350	(sugar)
58	0.310125	(garlic)
16	0.295150	(butter)
49	0.243250	(egg)
55	0.216350	(flour)
420	0.174950	(onion, garlic)
77	0.138500	(milk)
376	0.132525	(egg, sugar)
230	0.125325	(butter, sugar)
402	0.124475	(flour, sugar)
357	0.123125	(flour, egg)
211	0.122700	(flour, butter)
209	0.113800	(butter, egg)

Still using the Apriori principle, we get the following association rules in our dataset.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(allspice)	(sugar)	0.011425	0.314350	0.006175	0.540481	1.719362	0.002584	1.492105	0.423224
1	(apple)	(sugar)	0.028200	0.314350	0.017250	0.611702	1.945927	0.008385	1.765784	0.500212
2	(applesauce)	(sugar)	0.006925	0.314350	0.005050	0.729242	2.319841	0.002873	2.532334	0.572903
3	(avocado)	(onion)	0.010075	0.333200	0.006050	0.600496	1.802210	0.002693	1.669071	0.449656
4	(bacon)	(onion)	0.034250	0.333200	0.021450	0.626277	1.879584	0.010038	1.784211	0.484564
5	(banana)	(egg)	0.023025	0.243250	0.010375	0.450597	1.852404	0.004774	1.377405	0.471006
6	(banana)	(sugar)	0.023025	0.314350	0.012800	0.555917	1.768467	0.005562	1.543970	0.444779
7	(bean)	(garlic)	0.026625	0.310125	0.013750	0.516432	1.665238	0.005493	1.426635	0.410412
8	(bean)	(onion)	0.026625	0.333200	0.017200	0.646009	1.938804	0.008329	1.883666	0.497463
9	(beef)	(garlic)	0.051300	0.310125	0.026600	0.518519	1.671966	0.010691	1.432817	0.423634
10	(beef)	(onion)	0.051300	0.333200	0.040075	0.781189	2.344505	0.022982	3.047380	0.604481
11	(beef broth)	(garlic)	0.011675	0.310125	0.007150	0.612420	1.974751	0.003529	1.779954	0.499438
12	(beef broth)	(onion)	0.011675	0.333200	0.009450	0.809422	2.429237	0.005560	3.498827	0.595298
13	(blueberry)	(egg)	0.010125	0.243250	0.005250	0.518519	2.131628	0.002787	1.571712	0.536305
14	(blueberry)	(sugar)	0.010125	0.314350	0.007350	0.725926	2.309292	0.004167	2.501696	0.572766
15	(bread)	(butter)	0.023750	0.295150	0.010850	0.456842	1.547830	0.003840	1.297689	0.362545
16	(bread flour)	(sugar)	0.009800	0.314350	0.006725	0.686224	2.182995	0.003644	2.185161	0.547277
17	(bread flour)	(yeast)	0.009800	0.021700	0.006800	0.693878	31.975924	0.006587	3.195780	0.978314
18	(broccoli)	(garlic)	0.014525	0.310125	0.006675	0.459552	1.481830	0.002170	1.276488	0.329951
19	(broccoli)	(onion)	0.014525	0.333200	0.006825	0.469880	1.410203	0.001985	1.257827	0.295169
20	(buttermilk)	(butter)	0.021300	0.295150	0.010525	0.494131	1.674171	0.004238	1.393346	0.411453
21	(chive)	(butter)	0.012050	0.295150	0.005100	0.423237	1.433971	0.001543	1.222078	0.306327
22	(chocolate chip)	(butter)	0.014250	0.295150	0.007850	0.550877	1.866431	0.003644	1.569393	0.470929
23	(cream)	(butter)	0.109450	0.295150	0.047050	0.429877	1.456468	0.014746	1.236311	0.351926
24	(cream cheese)	(butter)	0.046300	0.295150	0.018675	0.403348	1.366586	0.005010	1.181341	0.281272
25	(egg)	(butter)	0.243250	0.295150	0.113800	0.467831	1.585063	0.042005	1.324486	0.487757
26	(egg yolk)	(butter)	0.011775	0.295150	0.007700	0.653928	2.215578	0.004225	2.036714	0.555188
27	(flour)	(butter)	0.216350	0.295150	0.122700	0.567137	1.921520	0.058844	1.628343	0.611981
28	(butter)	(flour)	0.295150	0.216350	0.122700	0.415721	1.921520	0.058844	1.341225	0.680398
29	(granulated sugar)	(butter)	0.027775	0.295150	0.014950	0.538254	1.823662	0.006752	1.526488	0.464556

Using the algorithm performing FP Growth, we reached the following results.

```
1 frequent_itemsets_fp1.sort_values(by = 'support', ascending = False).head(20)
```

	support	itemsets
11	0.333200	(onion)
16	0.314350	(sugar)
12	0.310125	(garlic)
0	0.295150	(butter)
1	0.243250	(egg)
2	0.216350	(flour)
275	0.174950	(onion, garlic)
20	0.138500	(milk)
154	0.132525	(egg, sugar)
149	0.125325	(butter, sugar)
164	0.124475	(flour, sugar)
162	0.123125	(flour, egg)
163	0.122700	(flour, butter)

We get the following association rules using the algorithm FP Growth.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(butter)	(sugar)	0.295150	0.314350	0.125325	0.424615	1.350770	0.032545	1.191636	0.368421
1	(onion, butter)	(garlic)	0.071875	0.310125	0.032375	0.450435	1.452430	0.010085	1.255311	0.335621
2	(butter, garlic)	(onion)	0.063850	0.333200	0.032375	0.507048	1.521752	0.011100	1.352667	0.366248
3	(egg)	(butter)	0.243250	0.295150	0.113800	0.467831	1.585063	0.042005	1.324486	0.487757
4	(egg)	(sugar)	0.243250	0.314350	0.132525	0.544810	1.733131	0.056059	1.506294	0.558982
5	(sugar)	(egg)	0.314350	0.243250	0.132525	0.421584	1.733131	0.056059	1.308315	0.616947
6	(butter, egg)	(sugar)	0.113800	0.314350	0.077100	0.677504	2.155255	0.041327	2.126075	0.604850
7	(butter, sugar)	(egg)	0.125325	0.243250	0.077100	0.615200	2.529087	0.046615	1.966608	0.691229
8	(egg, sugar)	(butter)	0.132525	0.295150	0.077100	0.581777	1.971123	0.037985	1.685345	0.567941
9	(egg, garlic)	(onion)	0.028875	0.333200	0.018100	0.626840	1.881272	0.008479	1.786900	0.482373
10	(flour)	(egg)	0.216350	0.243250	0.123125	0.569101	2.339572	0.070498	1.756212	0.730647
11	(egg)	(flour)	0.243250	0.216350	0.123125	0.506166	2.339572	0.070498	1.586871	0.756619
12	(flour)	(butter)	0.216350	0.295150	0.122700	0.567137	1.921520	0.058844	1.628343	0.611981
13	(butter)	(flour)	0.295150	0.216350	0.122700	0.415721	1.921520	0.058844	1.341225	0.680398
14	(flour)	(sugar)	0.216350	0.314350	0.124475	0.575341	1.830256	0.056465	1.614589	0.578866
15	(flour, egg)	(sugar)	0.123125	0.314350	0.091575	0.743756	2.366014	0.052871	2.675774	0.658416
16	(flour, sugar)	(egg)	0.124475	0.243250	0.091575	0.735690	3.024419	0.061296	2.863114	0.764522
17	(egg, sugar)	(flour)	0.132525	0.216350	0.091575	0.691002	3.193907	0.062903	2.536098	0.791843
18	(flour)	(egg, sugar)	0.216350	0.132525	0.091575	0.423272	3.193907	0.062903	1.504133	0.876544
19	(flour, butter)	(egg)	0.122700	0.243250	0.073125	0.595966	2.450013	0.043278	1.872985	0.674614

Both algorithms identify well items that are usually used together in recipes, however, it seems that APriori gets more range in regards to ingredients. When we verify the association rules, APriori identify a higher variety of items, whereas FP Growth identify many time the same products in different combinations. The decision of which one of them is better would be possible only analysing the necessities of the business.

As we are dealing with recipes, most of the rules that we may find here are likely obvious, because items cannot be combined randomly when cooking, however, we can still use this information in formatting marketing strategies, and understanding preferences of people who access this website.

Words Counting: 244.

6 Building a Dashboard to visualize the data

When it comes to the business we are exploring on our data, this is, using spaces in a culinary page to generate sales, and therefore income, some metrics are important to consider, the number of views on a certain page and the engagement of a user on the page. For instance, a recipes blog could sell space for advertising to grocery stores, that could explore this space to offer products related to the recipe that is being seen by the user. In this example, both factors listed above would increase the value generated for the grocery store when using the space.

In the case of our dataset, what we have in regards to these factors are n_reviews, rating and the reviews itself, that could be parsed to deliver a more accurate sentiment analysis than just rating. Watching this factors, we can analyse how some variables influence on the, such as minutes, n_steps and n_ingredients.

We will build a dashboard that deals with this variables, and that can deliver the information of how to generate value to the blog through recommendation systems explored before.

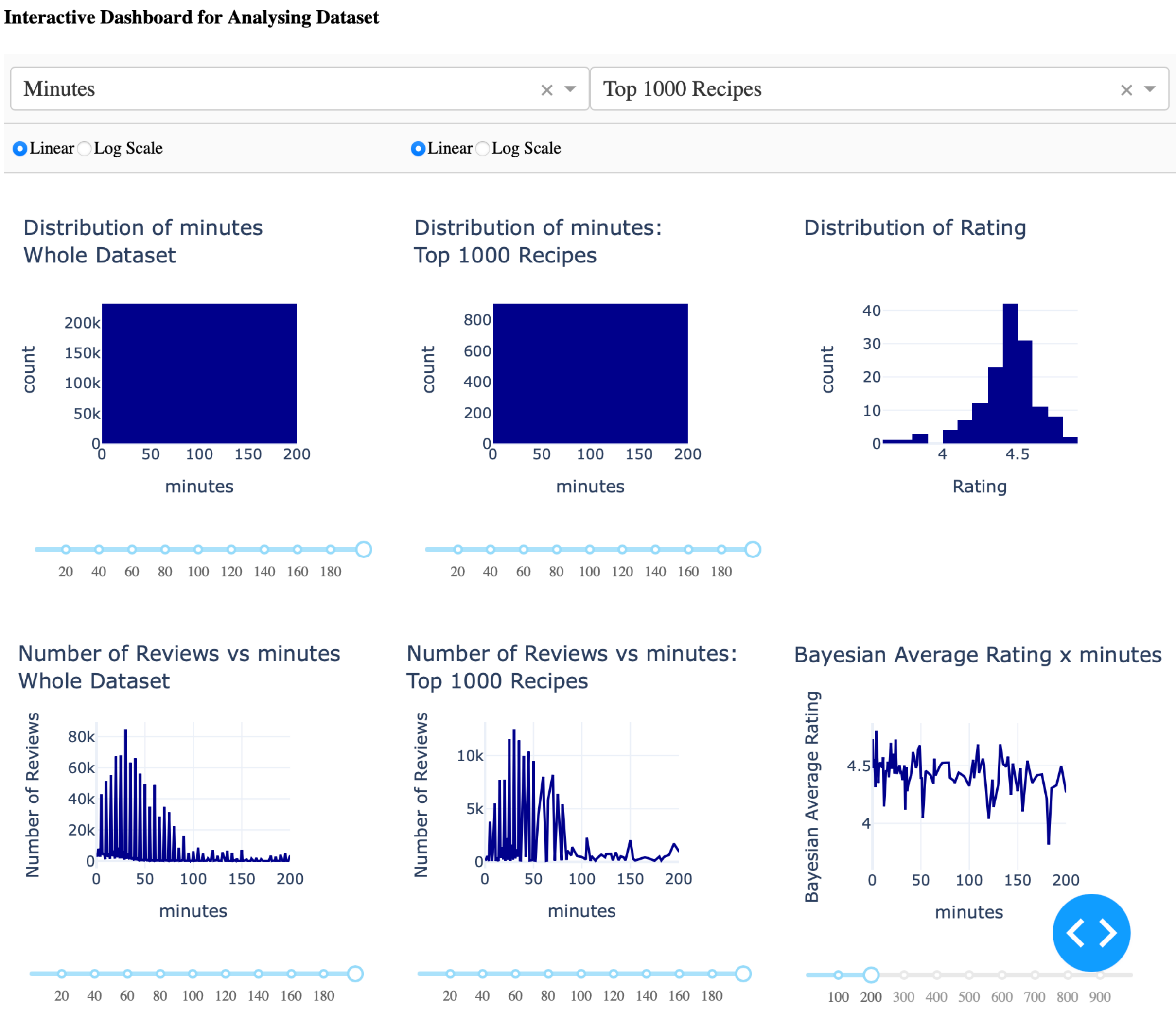
We will use Bayesian average as suggested by (Bengfort, 2017) that discussed its use and apply according to (Odabaşı, 2019) that shows a practical example how to calculate it using R.

The Bayesian average is calculated through the formula below, in which m is used for the prior value of average ratings and C is used for confidence in the prior values. Following the suggestion of (Odabaşı, 2019), I will use average of rating by recipe for m, and average number of reviews by recipe for C.

$$bayesianAverage = (C \times m + avgRating \times reviewCount) \div (C + reviewCount)$$

Given the initial ideas, I decided to create an interactive graph in which I can select among the variables stated, and visualize the distribution of the data, number of reviews vs variables, and Bayesian average, averaged by the variable vs the variable. Some features contain data varying widely, so it was decided to add the possibility to change between Linear Scale or Logarithmic Scale on the graph, making possible to observe better a wide range of values. It was also included on the graph the possibility of adjusting the length of the x-axis, so that it is possible to select parts of the graph that might be triggering in an analysis, and visualize it closer.

In regards to the target for the graph being people over 65, font sizes were increased, especially on titles, the design was decided to be kept simple, white background and dark blue colors, presenting high contrast, and not too much information on the graphs, just what is essential, in such a way that information is not overwhelming.



It is possible to verify for example that the majority of our reviews are given in recipes that take under 60 minutes to finish, approximately 8 steps, and no much more than 10 ingredients to get ready, therefore, spaces for advertising in recipes with these features are more likely to engage customers.

An observation in regards to the algorithm of the dashboards itself is that the library used to produce the graphs has an issue in regards to logarithm scales on histogram, and therefore, when changing to logarithm scale, no error will arise, however no graph will appear. This issue is being discussed in these links https://github.com/plotly/plotly.py/issues/2899, https://github.com/plotly/plotly.js/issues/6200.

Words counting: 550.

7 References

Bengfort, B., 2017. *District Insights.* [Online]
Available at: https://medium.com/district-data-labs/computing-a-bayesian-estimate-of-star-rating-means-651496a890ab
[Accessed 01 12 2023].

Cloud, S., 2023. *SaturnCloud.* [Online]
Available at: https://saturncloud.io/blog/how-to-select-rows-from-a-dataframe-based-on-list-values-in-a-column-in-pandas/#:~:text=To%20select%20rows%20from%20a%20DataFrame%20based%20on%20a%20list,to%20select%20the%20desired%20rows.
[Accessed 29 11 2023].

Coppola, D., 2023. *Statista.* [Online]
Available at: https://www.statista.com/statistics/678153/amazon-ad-cost/#:~:text=Since%202014%2C%20Amazon%27s%20annual%20advertising,dollars%20in%20the%20previous%20year
[Accessed 30 11 2023].

Leitch, J., 2020. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/building-a-recipe-recommendation-api-using-scikit-learn-nltk-docker-flask-and-heroku-bfc6c4bdd2d4
[Accessed 28 11 2023].

Li, S., 2019. *Kaggle.* [Online]
Available at: https://www.kaggle.com/dsv/783630
[Accessed 22 11 2023].

Mimi Polner, C. B., 2023. *Forbes Advisor.* [Online]
Available at: https://www.forbes.com/advisor/business/start-a-blog/
[Accessed 11 29 2023].

Montevirgen, K., 2023. *Brittanica Money.* [Online]
Available at: https://www.britannica.com/money/retailing
[Accessed 30 11 2023].

Naveen(NKK), 2023. *Spark by {Examples}.* [Online]
Available at: https://sparkbyexamples.com/pandas/conver-pandas-column-to-list/#
[Accessed 29 11 2023].

Odabaşı, E., 2019. *Medium.* [Online]
Available at: https://medium.com/@ertuodaba/web-scraping-with-python-and-calculating-bayesian-averages-with-sql-for-better-product-ranking-218dcbb75e4b
[Accessed 01 12 2023].