Configuration Manual

MSc Research Project
Data Analytics

Sunanda Pal
Student ID: x21195820

School of Computing
National College of Ireland

Supervisor: Vitor Horta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sunanda Pal
Student ID:	x21195820
Programme:	MSc. Data Analytics **Year:** 2023
Module:	MSc Research Project
Lecturer:	Vitor Horta
Submission Due Date:	18th August, 2023
Project Title:	Recommendation System for Food Dishes in Specific Restaurants Based on Sentiment Analysis
Word Count:	1563 **Page Count:** 15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sunanda Pal

Date: 17th August, 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sunanda Pal
Student ID: x21195820

1. Introduction

This research study adheres to a particular implementation setup, and the purpose of this manual is to provide guidance about the overall establishment of the configuration. This documentation offers in-depth insights into the software, hardware, and library setups employed during the project's development. Moreover, it elaborates on the programming approach and the steps required for executing the code.

2. System Configuration

This section describes the hardware and software specifications.

2.1 Hardware Configuration

The hardware specification is given below:

- **Windows Edition:** Microsoft Windows 10 Home Single Language
- **Processor:** AMD Ryzen 3 350U with Radeon Graphics 2.60 GHz
- **RAM:** 8.00 GB (5.94 GB usable)
- **System Type:** x64 based PC. 64-bit operating system.

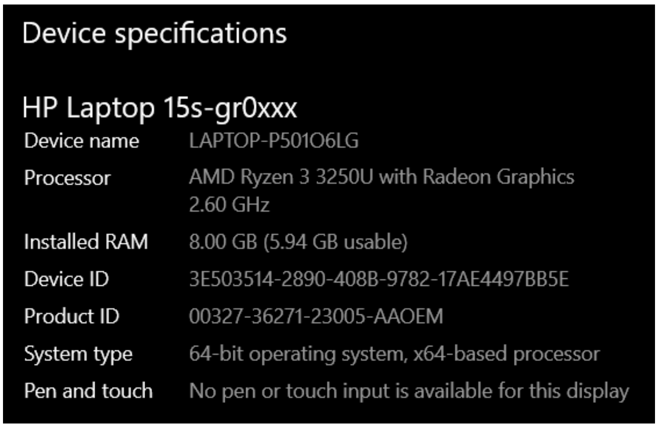

Figure 1: Device Specification

2.2Software Configuration

The software requirements of the study are given below:

- **Programming Language:** Python 3.10.7

- **IDE:** Jupyter Notebook

3. Project Implementation

This section describes the implementation steps of the project.

3.1 Programming Environment Set-up

The execution environment for implementing it is initiated by launching the Jupyter Notebook through the command prompt. The diagram below, on the left depicts the launch of Jupyter Notebook. Once, it is launched, a new tab called ‘Home’ opens in the browser which is shown in the diagram below on the right.

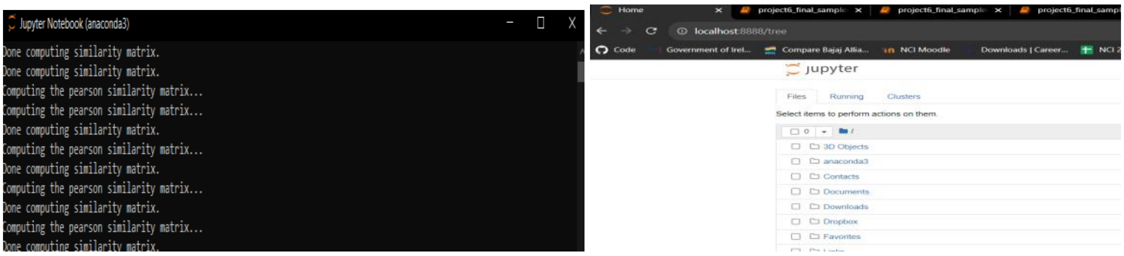

Figure 2: Execution environment: Jupyter Notebook launch (left) and Homepage of Jupyter (right)

3.2 Data Collection

The dataset utilized in the research is downloaded from Yelp.com[1] website. The Yelp website provides an openly accessible, versatile dataset sourced from real-world businesses, intended for both personal and academic use. This dataset is available in JSON format and contains around 6,990,280 reviews related to 150,346 businesses across 11 metropolitan cities. The dataset consists of six JSON files: business, reviews, user, checkin, tip, and photo. For our research purpose, only business, tip and review files are used. Data is downloaded in a zip format which is later extracted.

3.3Python Libraries

The libraries used in this study and their versions are listed below:

Table 1: Python Libraries and versions

Library	Version
seaborn[2]	0.12.1
pandas[3]	1.3.4

[1] https://www.yelp.com/dataset
[2] https://seaborn.pydata.org/
[3] https://pandas.pydata.org/

matplotlib[4]	3.6.2
json[5]	0.9.6
numpy[6]	1.22.4
plotly[7]	5.15.0
imageio[8]	2.9.0
folium[9]	0.14.0
scikit-surprise[10]	1.1.1
nltk[11]	3.8.1

All the libraries are installed in Jupyter Notebook using pip command.

3.4 Data Loading, EDA and Data Selection

Once all the necessary libraries are installed and imported, data is loaded. In this case, the original JSON files are stored in local directory and loaded first into the notebook. For the sake of simplicity, the JSON files are then converted to CSV. It is done in a python3 file named as "project1_readjson.ipynb". Then CSV files are loaded in another python3 file "project2_businessEDA_merged.ipynb" for EDA purpose and a dataframe is created with only necessary data from business, tip and review files. Here, data related to only restaurants in a particular city is chosen where review count is high. It is then saved into a csv file called "business_review_tip_merged.csv".

Business.json

```
business_df = pd.read_json('E:\\NCI Coursework\\SEM 2\\RIC\\Dataset\\yelp\\yelp_academic_dataset_business.json',lines= True)
business_df.head()
```

	business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open	
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-119.711197	5.0	7	0	{'ByAppointmentOr
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	-90.335695	3.0	15	1	{'BusinessAcceptsCr
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-110.880452	3.5	22	0	{'BikeParki 'BusinessAccep
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.955505	-75.155564	4.0	80	1	{'RestaurantsDeliver 'Outd
4	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA	18054	40.338183	-75.471659	4.5	13	1	{'BusinessAcceptsCr 'True',

Figure 3: Data loading of business.json

---

[4] https://pypi.org/project/matplotlib/
[5] https://docs.python.org/3/library/json.html
[6] https://numpy.org/
[7] https://plotly.com/python/getting-started/
[8] https://pypi.org/project/imageio/
[9] https://python-visualization.github.io/folium/
[10] https://surpriselib.com/#:~:text=Surprise%20is%20a%20Python%20scikit,perfect%20control%20over%20their%20experiments.
[11] https://www.nltk.org/

```
#json to csv
business_df.to_csv('yelp_business.csv')
tip_df.to_csv('yelp_tip.csv')
review_df.to_csv('yelp_review.csv')
```

Figure 4: Json files are converted to csv

```
# Checking cities where business is high
plt.subplot(1,2,1)
plotTopFreq(foods_df,"categories",10,"Top Business")
plt.subplot(1,2,2)
plotTopFreq(foods_df,"city",10,"Top cities")
```

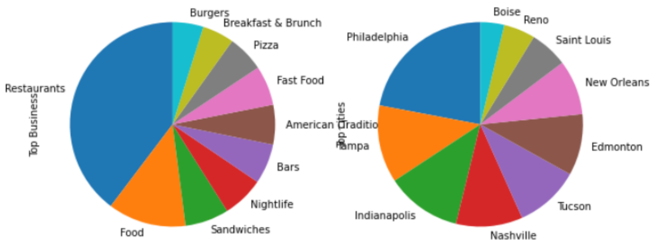

Figure 5: EDA of business types and cities

```
# Merging business_review with tip on 'business_id' and 'user_id' with all columns. Will merge nothing if not a match. Left-join
df_merged_final = df_merged.merge(tip[['business_id', 'user_id', 'text']],
                                  how='left',
                                  on=['business_id', 'user_id'])
```

```
df_merged_final.shape
```

(399866, 10)

```
df_merged_final.head()
```

	review_id	user_id	business_id	stars	review_text	city	categories	name	address	text
0	z0osLHDvXvzfm57D4DmD2Q	xVKE_HJ2pwUtTdLbL3pnCg	S2Ho8yLxhKAa26pBAm6rxA	3.0	Service was crappy, and food was mediocre. I ...	New Orleans	Cajun/Creole, Seafood, Restaurants, Breakfast ...	Creole House Restaurant & Oyster Bar	509 Canal St	NaN
1	tXHWJWnTdrraHGUqaPWj3g	zKAHSNzqvwsyoFCw3QpafA	S2Ho8yLxhKAa26pBAm6rxA	4.0	Enjoyed my fish out at a sidewalk table. A bi...	New Orleans	Cajun/Creole, Seafood, Restaurants, Breakfast ...	Creole House Restaurant & Oyster Bar	509 Canal St	NaN
2	IrZiB0XfNZjk8zfYDx_TPA	29UB_wmUldsxV2ZmrlZSg	S2Ho8yLxhKAa26pBAm6rxA	3.0	I was happy his was my first experience with N...	New Orleans	Cajun/Creole, Seafood, Restaurants, Breakfast ...	Creole House Restaurant & Oyster Bar	509 Canal St	NaN
3	_ZdwS4iEzJIVpy7-DjkEpA	IQU18Ke0zK8o4tPDirR07w	S2Ho8yLxhKAa26pBAm6rxA	5.0	Had breakfast with the family after a quick st...	New Orleans	Cajun/Creole, Seafood, Restaurants, Breakfast ...	Creole House Restaurant & Oyster Bar	509 Canal St	NaN
4	6_3e54OjnFnTxSgKrUZ21g	DbNKK25oOzfHxyfBHIOaDg	S2Ho8yLxhKAa26pBAm6rxA	5.0	The one thing I really wanted for breakfast wh...	New Orleans	Cajun/Creole, Seafood, Restaurants, Breakfast ...	Creole House Restaurant & Oyster Bar	509 Canal St	NaN

Figure 6: Merging records from business, review and tip

3.5 Data Pre-processing

Next, data pre-processing is done to prepare the review text data suitable for feature extraction. The output csv file of the second jupyter file is loaded and pre-processing steps are performed. A python3 file “project3_preprocess.ipynb” is used. The result is store in “token_pos_nolemm_df_new.csv” file. It has 11 columns where ‘business_id’ is restaurant id, ‘user_id’ is id of user, ‘review_id’ is unique id of review, ‘text’ is the user comment, ‘city’ is the city of the restaurant, ‘categories’ is the cuisine, ‘name’ is the restaurant name, ‘address’ is the restaurant location, ‘text_tokens’ is the tokens generated from ‘text’ and ‘ngrams’ is the 3-grams generated from the tokens.

```
merged_df.duplicated().sum() #no duplicates
```

```
0
```

```
#checking for null
merged_df['text'].isna()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
386998    False
386999    False
387000    False
387001    False
387002    False
Name: text, Length: 387003, dtype: bool
```

```
nan_values= merged_df[merged_df['text'].isna()]
nan_values    # no null values
```

business_id	user_id	review_id	stars	text	city	categories	name	address

Figure 7: Duplicate and null value check

```
# Set the value of N for N-grams
N = 3  # setting N to the desired value for the size of N-grams

# Generate N-grams from the 'text_tokens' column
merged_df['ngrams'] = merged_df['text_tokens'].apply(lambda tokens: list(ngrams(tokens, N)))
```

```
merged_df.head()
```

business_id	user_id	review_id	stars	text	city	categories	name	address	text_tokens	ngrams
0__F9fnKt8uioCKztF5Ww	002sVJCpSdFDqb6mCx9okg	i51UYC-axeOZAp8eyR3O-Q	1.0	located in the back of the catahoula hotel i t...	New Orleans	Cafes, Nightlife, Cocktail Bars, Peruvian, Res...	Piscobar	914 Union St	[located, back, catahoula, hotel, thought, fou...	[(located, back, catahoula), (back, catahoula,...
0__F9fnKt8uioCKztF5Ww	0G-QF457q_0Z_jKqh6xWiA	pF1BBNKDrQgfxLEOiZsyCg	5.0	i absolutely love this bar! even though i live...	New Orleans	Cafes, Nightlife, Cocktail Bars, Peruvian, Res...	Piscobar	914 Union St	[absolutely, love, bar, !, even, though, live,...	[(absolutely, love, bar), (love, bar, !), (bar...
0__F9fnKt8uioCKztF5Ww	0lgx-a1wAstiBDerGxXk2A	w2X-F8UhPaOVsOkeH2Xybw	4.0	so many hotel bars are soulless... just a spot...	New Orleans	Cafes, Nightlife, Cocktail Bars, Peruvian, Res...	Piscobar	914 Union St	[many, hotel, bars, soulless, ..., spot, unwin...	[(many, hotel, bars), (hotel, bars, soulless),...

Figure 8: Data-frame after tokenization, stop-word removal and n-grams

3.6 Feature Engineering

Feature engineering is one of the most crucial steps of the project. The data is loaded from the output file of the preprocessing step. In first phase, a food dictionary is created and stored in “food_dict_final” after refinement of food dishes. Then from the ‘text_tokens’ food items are extracted and stored in a separate column named ‘food_names’. Ngrams are also filtered and stored in column named ‘filtered_ngrams’. The name of the python3 file used here is “project4_foodDictionaryFoodExtraction.ipynb”.

```
food_dictionary
```

```
['branch_water',
 'pigweed',
 'pistachio_nut',
 'limeade',
 'spotted_dick',
 'serviceberry',
 'lunch',
 'garlic',
 'veggie',
 'brewage',
 'fillet',
 'fruit_punch',
 'sirloin_tip',
 'peanut_oil',
 'bock',
 'corn_gluten',
 'pork_tenderloin',
 'rechewed_food',
 'rock_candy',
```

Figure 9: Food_dictionary creation

```
len(food_dictionary)
```

```
3583
```

```
food_dictionary1 = ['sangaree','costmary', 'pie_shell', 'maconnais', 'wedding_cake', 'buttermilk', 'cachou', 'gin', 'pinwheel_ro]
```

```
food_dictionary2= ['flame_tokay', 'coq_au_vin', 'dropped_egg', 'merlot', 'picnic_shoulder', 'red_delicious', 'sauce_louis', 'sai]
```

```
food_dictionary3= [ 'pine_nut', 'frozen_pudding', 'shandy', 'doughnut', 'frankfurter', 'cruller', 'pork_sausage', 'hallah', 'walc
```

```
food_dictionary4= ['salad', 'falafel', 'sauce_albert', 'baking-powder_biscuit', 'roast_beef', 'brisket', 'granadilla', 'spanish_c
```

```
food_dictionary5= ['muscadelle', 'fillet', 'mahimahi', 'drop_biscuit', 'pfannkuchen', 'coconut', 'camembert', 'wild_rice', 'mines
```

```
food_dict_final= food_dictionary1 + food_dictionary2 +food_dictionary3 +food_dictionary4 + food_dictionary5
```

```
len(food_dict_final)
```

```
2907
```

Figure 10: Food_dictionary after refinement

After those steps, a new csv file is created named “filtered_ngrams_dict_new.csv”. That is used in another python3 file called “project5_posExtraction” to do the later steps of feature engineering such as POS tagging, making {food: description} where ‘food’ is dish and ‘description’ is word describing opinion of user and then getting positive and negative sentiment score from sentiment analysis. “filtered_pos_tags” contains the POS tags, “food_descriptions” contains the food and opinion pair, “sentiment_scores” has positive and negative score for each food opinion and finally “average_scores” has average of the sentiment scores for each food item.

```
# Perform POS tagging on the filtered N-grams
def pos_tag_ngrams(ngrams):
    tagged_ngrams = []
    for ngram in ngrams:
        tagged_ngram = nltk.pos_tag(ngram)
        tagged_ngrams.append(tagged_ngram)
    return tagged_ngrams
```

```
filtered_df['filtered_pos_tags'] = filtered_df['filtered_ngrams'].apply(pos_tag_ngrams)
```

```
filtered_df.head()
```

	business_id	user_id	review_id	stars	text	city	categories	name	address	text_tokens	r
0	-0__F9fnKt8uioCKztF5Ww	0G-QF457q_0Z_jKqh6xWiA	pF1BBNKDrQgfxLEOiZsyCg	5.0	i absolutely love this bar! even though i live...	New Orleans	Cafes, Nightlife, Cocktail Bars, Peruvian, Res...	Piscobar	914 Union St	['absolutely', 'love', 'bar', '!', 'even', 'th...	[('abs 'love ('love
1	-0__F9fnKt8uioCKztF5Ww	1DjkPbctTZ4SV_MS3TaeTQ	MKLDHCphgJ2SCTsLwF7IWg	5.0	what a beautiful way to make use of gorgeous o...	New Orleans	Cafes, Nightlife, Cocktail Bars, Peruvian, Res...	Piscobar	914 Union St	['beautiful', 'way', 'make', 'use', 'gorgeous'...	[('be 'r

Figure 11: POS tagging

```
def extract_food_descriptions(grams, food_tokens):
    food_descriptions = []
    for gram in grams:
        food_token = next((token for token, tag in gram if token in food_tokens), None)
        if food_token:
            description = next((token for token, tag in gram if tag in ['NN', 'JJ', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'] and
            if description:
                food_descriptions.append({food_token: description})
    return food_descriptions

filtered_df['food_descriptions'] = filtered_df.apply(lambda row: extract_food_descriptions(row['filtered_pos_tags'], row['food_tc
```

Figure 12: Food-description extraction

```python
# Define the function to get positive and negative sentiment scores for a word
def get_sentiment_scores(word):
    synsets = list(swn.senti_synsets(word))
    if not synsets:
        return None, None

    # Consider the first synset as it generally represents the most common usage of the word
    synset = synsets[0]
    pos_score = synset.pos_score()
    neg_score = synset.neg_score()

    return pos_score, neg_score
```

```python
def process_row(row):
    sentiment_scores = []
    for pair in row['food_descriptions']:
        key = list(pair.keys())[0]
        value = list(pair.values())[0]
        pos_score, neg_score = get_sentiment_scores(value)
        sentiment_scores.append({ 'food': key, 'description': value, 'pos': pos_score, 'neg': neg_score})
    return sentiment_scores
```

```python
# Apply the processing function to each row
filtered_df['sentiment_scores'] = filtered_df.apply(process_row, axis=1)
```

Figure 13: Sentiment score generation

```python
filtered_df.to_csv('filtered_sentiment_scores_new.csv')
```

```python
filtered_df.head()
```

egories	name	address	text_tokens	ngrams	food_tokens	filtered_ngrams	filtered_pos_tags	food_descriptions	sentiment_scores	average_scores
Cafes, lightlife, Cocktail Bars, eruvian, Res...	Piscobar	914 Union St	['absolutely', 'love', 'bar', '!', 'even', 'th...	[('absolutely', 'love', 'bar'), ('love', 'bar'...	[coffee, espresso]	[(., delicious, coffee), (delicious, coffee, d...	[[(., .), (delicious, JJ), (coffee, NN)], [(de...	[{'coffee': 'delicious'}, {'coffee': 'deliciou...	[{'food': 'coffee', 'description': 'delicious'...	[{'food': 'coffee', 'pos': 0.0, 'neg': 0.0}, {...
Cafes, lightlife, Cocktail Bars, eruvian, Res...	Piscobar	914 Union St	['beautiful', 'way', 'make', 'use', 'gorgeous'...	[('beautiful', 'way', 'make'), ('way', 'make',...	[popcorn, jambalaya]	[(rooftop, (, popcorn), ((, popcorn, jambalaya...	[[(rooftop, NN), ((, (), (popcorn, JJ)], [((, ...	[{'popcorn': 'rooftop'}, {'popcorn': 'jambalay...	[{'food': 'popcorn', 'description': 'rooftop',...	[{'food': 'popcorn', 'pos': 0.0, 'neg': 0.0}, ...
Cafes, lightlife, Cocktail Bars, eruvian, Res...	Piscobar	914 Union St	['hidden', 'gem', 'great', 'cocktails', ',', '...	[('hidden', 'gem', 'great'), ('gem', 'great', ...	[banana, smoothie]	[(:, order, banana), (order, banana, smoothie)...	[[(:, :), (order, NN), (banana, NN)], [(order,...	[{'banana': 'order'}, {'banana': 'order'}, {'b...	[{'food': 'banana', 'description': 'order', 'p...	[{'food': 'banana', 'pos': 0.20833333333333334...
Cafes, lightlife, Cocktail Bars, eruvian, Res...	Piscobar	914 Union St	['cool', 'spot', 'regardless', '"re", 'inside'...	[('cool', 'spot', 'regardless'), ('spot', 'reg...	[cocktail]	[(damn, good, cocktail), (good, cocktail, .), ...	[[(damn, RB), (good, JJ), (cocktail, NN)], [(g...	[{'cocktail': 'good'}, {'cocktail': 'good'}]	[{'food': 'cocktail', 'description': 'good', '...	[{'food': 'cocktail', 'pos': 0.5, 'neg': 0.0}]

Figure 14: Output of file "project5_posExtraction

Once, those mentioned steps are done, data-frame is stored in a file named 'filtered_sentiment_score_new.csv'. There are few more steps of feature engineering left which is done in the beginning of every 'project6 files which are the files basically created for applying collaborative filtering models. From the data loaded from 'filtered_sentiment_score_new.csv' file, only four attributes 'user_id', 'business_id', 'stars', 'average_scores' are taken and stored in rating_df dataframe.

```
# Create an empty DataFrame 'rating_df'
rating_df = pd.DataFrame(columns=['user_id', 'business_id', 'stars', 'average_scores'])
```

```
rating_df['user_id']= filtered_df['user_id']
rating_df['business_id']= filtered_df['business_id']
rating_df['stars']= filtered_df['stars']
rating_df['average_scores']= filtered_df['average_scores']
```

```
rating_df.head()
```

	user_id	business_id	stars	average_scores
0	002sVJCpSdFDqb6mCx9okg	-0__F9fnKt8uioCKztF5Ww	1.0	[{'food': 'gem', 'pos': 0.020833333333333332, ...
1	0G-QF457q_0Z_jKqh6xWiA	-0__F9fnKt8uioCKztF5Ww	5.0	[{'food': 'delicious', 'pos': 0.0, 'neg': 0.0}...
2	0Igx-a1wAstiBDerGxXk2A	-0__F9fnKt8uioCKztF5Ww	4.0	[{'food': 'vintage', 'pos': 0.3333333333333333...
3	1DjkPbctTZ4SV_MS3TaeTQ	-0__F9fnKt8uioCKztF5Ww	5.0	[{'food': 'drink', 'pos': 0.09375, 'neg': 0.0}...
4	1EwRVVOdeWca3Of2fiSV1Q	-0__F9fnKt8uioCKztF5Ww	5.0	[{'food': 'food', 'pos': 0.0, 'neg': 0.0625}, ...

Figure 15: Necessary data loaded in 'rating_df' data-frame

As 'average_scores' has information about food iem, positive and negative score, they are first unpacked and stored into separate columns. The attribute 'stars' is transformed through 'min_max' normalization and stored in 'normalized_stars'.

```
# Perform Min-Max normalization
min_rating = 0 # min rating can be 0
max_rating = rating_df['stars'].max() # max rating is 5

rating_df['normalized_stars'] = (rating_df['stars'] - min_rating) / (max_rating - min_rating)
```

```
rating_df.head()
```

	user_id	business_id	stars	food	total_score	normalized_stars
0	002sVJCpSdFDqb6mCx9okg	-0__F9fnKt8uioCKztF5Ww	1	gem	-0.208333	0.2
1	002sVJCpSdFDqb6mCx9okg	-0__F9fnKt8uioCKztF5Ww	1	drink	-0.062500	0.2
2	002sVJCpSdFDqb6mCx9okg	-0__F9fnKt8uioCKztF5Ww	1	bit	0.125000	0.2
3	002sVJCpSdFDqb6mCx9okg	-0__F9fnKt8uioCKztF5Ww	1	beverages	0.125000	0.2
4	0G-QF457q_0Z_jKqh6xWiA	-0__F9fnKt8uioCKztF5Ww	5	delicious	0.000000	1.0

Figure 16: Min-max transformation of rating 'stars'

Next, for every record a tuple of 'food' and 'business_is' is created to generate 'restaurant_food_pair' and a 'final_rating' is generated by taking average of 'normalized_stars' and and 'total_score'. Now, the data is prepared to be utilized in model implementation.

3.7 Sampling of Data

Due to limited computational resource, the experiment is carried out in two ways. In one-way, whole data is taken and in other way, only a small part of the data is taken. Hence, sampling is done in file 'project6_sampled_CFknnBasic.ipynb' with only 10000 records and stored in 'sampled_df.csv' for further utilization.

Sampling of Data -Because of Memory Issue

```
# Sample 10000 rows of the 'filtered_df' DataFrame
sampled_df = filtered_df.sample(n=10000, random_state=42)
```

```
filtered_df.shape
```

```
(2286259, 6)
```

```
sampled_df.shape
```

```
(10000, 6)
```

```
sampled_df.to_csv('sampled_df.csv')
```

Figure 17: Data Sampling

3.8 Train-test Split

In both experimental approaches, the data is randomly split into training and test data in 80:20 ratio. AS for collaborative filtering models surprise library is used, the original data is first loaded into surprise dataset and then the split is done. A ‘reader’ object is created too to specify the rating scale which is 0-1 in this case.

Train test split

```
# Create a Reader object to specify the rating scale (here, the total_score ranges from 0 to 1)
reader = Reader(rating_scale=(0, 1))
```

```
# Load the data from the DataFrame into the Surprise Dataset
data = Dataset.load_from_df(sampled_df[['user_id', 'restaurant_food_pair', 'final_rating']], reader)
```

```
# Split the data into train and test sets
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

Figure 18: Train-test split

3.9 Model Implementation

Total four models are implemented. On sampled data, KNNBasic and KNNWithMeans and on whole data, SVD and NMF are applied.

3.9.1 KNNBasic

First KNNBasic() with default setting is applied. Then 5-fold cross-validation and hyper parameter tunning with GridSearchCV is applied as well. The python3 file name is “project6_sampled_CFknnBasic.ipynb”.

KNNBasic

```
# Create the KNNBasic algorithm
algorithm = KNNBasic() #msd similarity

# Train the algorithm on the trainset
algorithm.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.

<surprise.prediction_algorithms.knns.KNNBasic at 0x2b3b57d69d0>
```

```
predictions = algorithm.test(testset)
```

Figure 19: KNNBasic() with default setting

cross-validation

```
# Perform cross-validation with the chosen algorithm
cv_results = cross_validate(algorithm, data, measures=['rmse', 'mae'], cv=5, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1539  0.1532  0.1504  0.1513  0.1504  0.1518  0.0014
MAE (testset)     0.1175  0.1190  0.1146  0.1166  0.1148  0.1165  0.0017
Fit time          1.85    1.87    1.80    1.80    1.78    1.82    0.03
Test time         0.04    0.03    0.04    0.05    0.04    0.04    0.01
```

```
# Get the average RMSE and MAE across the folds
```

Figure 20: KNNBasic() with 5-fold cross-validation

Hyperparameter tunning

```
# Define the parameter grid to search
param_grid = {
    'k': range(10,50,1),              # Number of neighbors to consider
    'min_k': [1, 3, 5],              # Minimum number of neighbors to consider
    'sim_options': {
        'name': ['cosine', 'pearson', 'pearson_baseline', 'msd'],   # Similarity metric to use ('cosine' or 'pearson')
        'user_based': [True, False]      # User-based or item-based (True or False)
    }
}
```

```
from surprise.model_selection import GridSearchCV
```

```
# Perform GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=5, n_jobs=-1)
grid_search.fit(data)
```

Figure 21: KNNBasic() with 5-fold cross-validation and hyper parameter tunning

3.9.2 KNNWithMeans

KNNWithMeans() is applied on the training data obtained from sampled dataset. After running with default setting, 5-fold cross validation and optimization through hyperparameter tunning is applied. It is executed in “project6_sampled_CFknnWithMeans.ipynb” python file.

KNNWithMeans

```
# Create the KNNWithMean algorithm
algorithm = KNNWithMeans()

# Train the algorithm on the trainset
algorithm.fit(trainset)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.

<surprise.prediction_algorithms.knns.KNNWithMeans at 0x15719b92580>
```

```
predictions = algorithm.test(testset)
```

Figure 22: KNNWithMeans() with default values

cross-validation

```
# Perform cross-validation with the chosen algorithm
cv_results = cross_validate(algorithm, data, measures=['rmse', 'mae'], cv=5, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1527  0.1531  0.1562  0.1560  0.1491  0.1534  0.0026
MAE (testset)     0.1165  0.1163  0.1191  0.1207  0.1148  0.1175  0.0021
Fit time          2.03    2.00    1.96    1.92    1.86    1.95    0.06
Test time         0.04    0.03    0.03    0.04    0.03    0.03    0.00
```

Figure 23: KNNWithMeans() with 5-fold cross-validation

Hyperparameter tunning

```
# Define the parameter grid to search
param_grid = {
    'k': range(10,50,1),              # Number of neighbors to consider
    'min_k': [1, 3, 5],              # Minimum number of neighbors to consider
    'sim_options': {
        'name': ['cosine', 'pearson', 'pearson_baseline', 'msd'],   # Similarity metric to use ('cosine' or 'pearson')
        'user_based': [True, False]      # User-based or item-based (True or False)
    }
}
```

```
from surprise.model_selection import GridSearchCV
```

```
# Perform GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(KNNWithMeans, param_grid, measures=['rmse', 'mae'], cv=5, n_jobs=-1)
grid_search.fit(data)
```

Figure 24: KNNWithMeans() with 5-fold cross-validation and hyperparameter tunning

3.9.3 SVD

On the complete dataset obtained after feature engineering, SVD algorithm is applied. In this case also first the basic version, then 5-fold cross validation and finally hyperparameter tunning using GridSearchCV with cross-validation is implemented. The file executed here is “project6_CFsvd_full.ipynb”.

SVD

```
# Use the SVD algorithm to build the model and train it on the training set
model1 = SVD()
model1.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x212a27132b0>
```

```
# Make predictions on the test set
predictions1 = model1.test(testset)
```

Figure 25: SVD() with default values

SVD - 5 cross fold- RMSE and MAE

```
# Choose the collaborative filtering algorithm (e.g., Singular Value Decomposition - SVD)
algorithm = SVD()

# Perform cross-validation with the chosen algorithm
cv_results = cross_validate(algorithm, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Get the average RMSE and MAE across the folds
average_rmse = cv_results['test_rmse'].mean()
average_mae = cv_results['test_mae'].mean()

print("Average RMSE:", average_rmse)
print("Average MAE:", average_mae)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1457  0.1465  0.1464  0.1464  0.1463  0.1463  0.0003
MAE (testset)     0.1119  0.1126  0.1125  0.1124  0.1125  0.1124  0.0002
Fit time          48.21   45.28   45.45   51.17   47.71   47.56   2.15
Test time         2.58    2.46    1.70    2.57    2.50    2.36    0.34
```

Figure 26: SVD() with 5-fold cross-validation

SVD- crossfold - Hyper parameter tunning - GridSearchCV

```
from surprise.model_selection import GridSearchCV
```

```
param_grid = {
    'n_epochs': [5, 10, 15],          # Number of iterations of the optimization algorithm
    'lr_all': [0.002, 0.005, 0.01],   # Learning rate for all parameters
    'reg_all': [0.1, 0.2, 0.4, 0.6]       # L2 regularization term for all parameters
}
```

```
import joblib

joblib.parallel_backend('loky')
# Use the SVD class instead of the SVD object
algorithm_class = SVD

# Perform GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(algorithm_class, param_grid, measures=['rmse', 'mae'], cv=5, n_jobs=-1)
```

```
grid_search.fit(data)  # Use the original 'data', not the trainset
```

Figure 27: SVD() with 5-fold cross-validation and hyperparameter tunning

3.9.4 NMF

A similar approach is followed for NMF as well. It is executed in “project6_CFnmf_full.ipynb”. Figure 28 shows the basic nmf() model, whereas fig 29 shows cross-validation on it and fig 30 depicts the optimization performed through hyperparameter tunning.

NMF

```
# Use the NMF algorithm to build the model and train it on the training set
model1 = NMF()
model1.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.NMF at 0x1c4838f9ac0>
```

```
# Make predictions on the test set
predictions1 = model1.test(testset)
```

Figure 28: NMF() with default setting

NMF - 5 cross fold- RMSE and MAE

```
# Choose the collaborative filtering algorithm NMF
algorithm = NMF()

# Perform cross-validation with the chosen algorithm
cv_results = cross_validate(algorithm, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Get the average RMSE and MAE across the folds
average_rmse = cv_results['test_rmse'].mean()
average_mae = cv_results['test_mae'].mean()

print("Average RMSE:", average_rmse)
print("Average MAE:", average_mae)
```

```
Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.1374  0.1381  0.1370  0.1379  0.1379  0.1377  0.0004
MAE (testset)     0.1057  0.1061  0.1054  0.1059  0.1059  0.1058  0.0002
Fit time          123.70  122.80  121.47  123.38  121.35  122.54  0.97
Test time         4.65    4.30    2.63    4.42    4.30    4.06    0.73
```

Figure 29: NMF() with 5-fold cross-validation

NMF- crossfold - Hyper parameter tunning - GridSearchCV

```
from surprise.model_selection import GridSearchCV
```

```
# Define the parameter grid to search
param_grid = {
    'n_factors': [10, 20, 30],  # Number of latent factors
    'reg_pu': [0.02, 0.05, 0.1],  # Regularization term for user factors
    'reg_qi': [0.02, 0.05, 0.1]   # Regularization term for item factors
}
```

```
import joblib

joblib.parallel_backend('loky')

# Perform GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(NMF, param_grid, measures=['rmse', 'mae'], cv=5, n_jobs=-1)
```

```
grid_search.fit(data)  # Use the original 'data', not the trainset
```

Figure 30: NMF() with 5-fold cross-validation and hyperparameter tunning

3.10 Results and Evaluation

For every model, results are obtained in terms of RMSE and MAE. Those values are compared for every model to chose the best one in each case.

3.10.1 KNNBasic

The RMSE and MSE value of the KNNBasic model trained on the best parameters are shown in the below diagram. RMSE value achieved is 0.1517 and MAE achieved is 0.1164.

```
# Get the best RMSE and MAE scores along with the best hyperparameters
best_rmse = grid_search.best_score['rmse']
best_mae = grid_search.best_score['mae']
best_params = grid_search.best_params['rmse']  # or 'mae' for the best hyperparameters

print("Best RMSE:", best_rmse)
print("Best MAE:", best_mae)
print("Best Hyperparameters:", best_params)
```

```
Best RMSE: 0.1517896483810298
Best MAE: 0.11649174765760958
Best Hyperparameters: {'k': 10, 'min_k': 1, 'sim_options': {'name': 'cosine', 'user_based': True}}
```

Figure 31: Result of hyperparameter tunned KNNBasic

3.10.2 KNNWithMeans

The diagram blow shows the best RMSE and MAE value obtained for optimized KNNWithMeans model. They are 0.1532 and 0.1173 respectively and shown in fig 32.

```
# Get the best RMSE and MAE scores along with the best hyperparameters
best_rmse = grid_search.best_score['rmse']
best_mae = grid_search.best_score['mae']
best_params = grid_search.best_params['rmse']  # or 'mae' for the best hyperparameters

print("Best RMSE:", best_rmse)
print("Best MAE:", best_mae)
print("Best Hyperparameters:", best_params)
```

```
Best RMSE: 0.15324264498628673
Best MAE: 0.11737935424312862
Best Hyperparameters: {'k': 10, 'min_k': 1, 'sim_options': {'name': 'cosine', 'user_based': True}}
```

Figure 32: Result of hyperparameter tunned KNNWithMeans

3.10.3 SVD

For SVD also, the best RMSE and MAE value is obtained for optimized model with the best combination of hyperparameters. The diagram below shows the result of RMSE and MAE which are 0.1272 and 0.0957 respectively.

```
# Get the best RMSE and MAE scores along with the best hyperparameters
best_rmse = grid_search.best_score['rmse']
best_mae = grid_search.best_score['mae']
best_params = grid_search.best_params['rmse']  # or 'mae' for the best hyperparameters

print("Best RMSE:", best_rmse)
print("Best MAE:", best_mae)
print("Best Hyperparameters:", best_params)
```

```
Best RMSE: 0.1272364609501841
Best MAE: 0.09571371173210905
Best Hyperparameters: {'n_epochs': 15, 'lr_all': 0.01, 'reg_all': 0.6}
```

Figure 33: Result of hyperparameter tunned SVD

3.10.4 NMF

Optimized model of NMF has shown best value for RMSE and MAE for the best combination of hyperparameters. The diagram below shows the result of it. RMSE value obtained is 0.1304 and MAE value obtained is 0.0967.

```
# Get the best RMSE and MAE scores along with the best hyperparameters
best_rmse = grid_search.best_score['rmse']
best_mae = grid_search.best_score['mae']
best_params = grid_search.best_params['rmse']  # or 'mae' for the best hyperparameters

print("Best RMSE:", best_rmse)
print("Best MAE:", best_mae)
print("Best Hyperparameters:", best_params)
```

```
Best RMSE: 0.1304115176563202
Best MAE: 0.09673086844157211
Best Hyperparameters: {'n_factors': 30, 'reg_pu': 0.02, 'reg_qi': 0.02}
```

Figure 34: Result of hyperparameter tunned NMF