Configuration Manual

MSc Research Project
Data Analytics

Akhil Bharat Sisal
Student ID: x21214638

School of Computing
National College of Ireland

Supervisor: Dr Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Akhil Bharat Sisal
Student ID	x21214638
Programme	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Christian Horn
Submission Due Date:	14/12/2023
Project Title:	Sentiment Analysis Over Yelp Dataset
Word Count:	1183
Page Count:	29

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	Akhil Bharat Sisal
Date	13/12/23

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

Akhil Bharat Sisal
Student ID: x21214638

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and its effects on a specific setting. A glimpse of the source for Exploratory Data analysis for is done followed by sentiment analysis, data preprocessing and cleaning and vectorization after that all the algorithms are created, and Evaluations is also supplied. After that recommendatory system is build and put together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating environment configuration provided in Section 2.
Information about data collection is detailed in Section 3. Exploratory Data Analysis is done in Section 4. Sentiment Analysis is included in Section 5. In section 6, the Data Clenaing is described. Section 7 provides details of Tokenisation. Details well about models that were created and tested are provided in Section 8. How the results are calculated and shown is described in Section 9.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. Dell Desktop-J1UJ4D9, Windows 11 operating system, 16GB RAM, i5 intel core processor, 1TB memory.

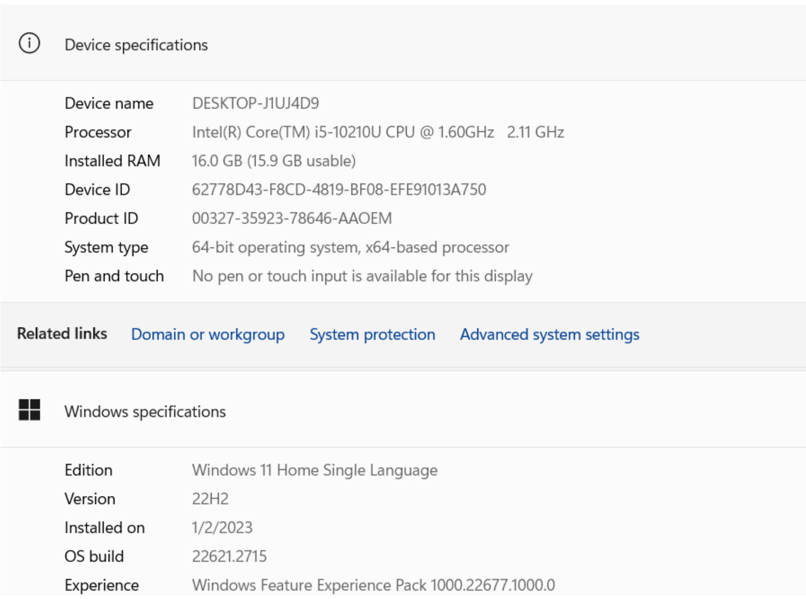


Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The jupyter notebook comes with Anaconda 3, run the jupyter notebook from startup. This will open jupyter notebook in web browser. The web browser will show the folder structure of the system, move to the folder where the code file is located. Open the code file from the folder and to run the code, go to Kernel menu and Run all cells.

3 Data Gathering

The dataset is collected from https://data.world/brianray/yelp-reviewsfor tabular data. The data is a detailed dump of Yelp reviews, businesses, users, and checkins for the Phoenix, AZ metropolitan area.

4 Exploratory Data Analysis

Figure 2 includes a list of every Python library necessary to complete the project.

```
# importing the necessary packages
import pandas as pd
import numpy as np
import nltk
nltk.download('vader_lexicon')
nltk.download('punkt')
nltk.download('stopwords')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from sklearn.preprocessing import MinMaxScaler
import string
import re
from wordcloud import WordCloud
from datetime import datetime
from tqdm import tqdm
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier, NearestNeighbors
from sklearn.cluster import DBSCAN, KMeans
from tqdm import tqdm
from surprise import Reader, Dataset
from surprise.model_selection import cross_validate
from surprise import KNNBasic, KNNWithMeans, SVD
from surprise.accuracy import rmse
from surprise import accuracy
from surprise.model_selection import GridSearchCV
import tensorflow as tf
import  transformers
from transformers import BertTokenizer
```

Figure 2: Necessary Python libraries

The Figure 3 represents the block of code to import data as pandas dataframe and print top 10 rows of the data.

```
# data import
data= pd.read_csv('yelp_training_set_review.csv')
```

```
data
```

	Unnamed: 0	business_blank	business_categories	business_city	business_full_address	business_id	business_latitude	business_longitude
0	0	False	Breakfast & Brunch; Restaurants	Phoenix	6106 S 32nd St\nPhoenix, AZ 85042	9yKzy9PApeiPPOUJEtnvkg	33.390792	-112.012504
1	1	False	Italian; Pizza; Restaurants	Phoenix	4848 E Chandler Blvd\nPhoenix, AZ 85044	ZRJwVLyzEJq1VAihDhYiow	33.305607	-111.978758
2	2	False	Middle Eastern; Restaurants	Tempe	1513 E Apache Blvd\nTempe, AZ 85281	6oRAC4uyJCsJl1X0WZpVSA	33.414345	-111.913031
3	3	False	Active Life; Dog Parks; Parks	Scottsdale	5401 N Hayden Rd\nScottsdale, AZ 85250	_1QQZuf4zZOyFCvXc0o6Vg	33.522945	-111.907886
4	4	False	Tires; Automotive	Mesa	1357 S Power Road\nMesa, AZ 85206	6ozycU1RpktNG2-1BroVtw	33.391027	-111.684482

Figure 3: Data import

As seen in Figure 4, the information of the data.

```
data.info()  #checking data information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 229907 entries, 0 to 229906
Data columns (total 32 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Unnamed: 0                229907 non-null  int64  
 1   business_blank            229907 non-null  bool   
 2   business_categories       229130 non-null  object 
 3   business_city             229907 non-null  object 
 4   business_full_address     229907 non-null  object 
 5   business_id               229907 non-null  object 
 6   business_latitude         229907 non-null  float64
 7   business_longitude        229907 non-null  float64
 8   business_name             229907 non-null  object 
 9   business_neighborhoods    0 non-null       float64
 10  business_open             229907 non-null  bool   
 11  business_review_count     229907 non-null  int64  
 12  business_stars            229907 non-null  float64
 13  business_state            229907 non-null  object 
 14  business_type             229907 non-null  object 
 15  cool                      229907 non-null  int64  
 16  date                      229907 non-null  object 
 17  funny                     229907 non-null  int64  
 18  review_id                 229907 non-null  object 
 19  reviewer_average_stars    229907 non-null  float64
 20  reviewer_blank            229907 non-null  bool   
 21  reviewer_cool             229907 non-null  int64  
 22  reviewer_funny            229907 non-null  int64  
 23  reviewer_name             215879 non-null  object 
 24  reviewer_review_count     229907 non-null  int64  
 25  reviewer_type             229907 non-null  object 
 26  reviewer_useful           229907 non-null  int64  
 27  stars                     229907 non-null  int64  
 28  text                      229901 non-null  object 
 29  type                      229907 non-null  object 
 30  useful                    229907 non-null  int64  
 31  user_id                   229907 non-null  object 
dtypes: bool(3), float64(5), int64(10), object(14)
memory usage: 51.5+ MB
```

Figure 4: Data information

In figure 5, the code to generate data subset of 1 lakh records.

```
data= data.sample(100000).reset_index()
data.head()
```

	index	Unnamed: 0	business_blank	business_categories	business_city	business_full_address	business_id	business_latitude	business_longitu
0	144486	144486	False	Gluten-Free; Pizza; Vegan; Restaurants	Phoenix	53 West Thomas Road\nPhoenix, AZ 85013	ShI6PtJERnowSJSC4IHbYQ	33.479992	-112.076
1	49118	49118	False	Food; Farmers Market	Phoenix	4700 E Warner Rd\nPhoenix, AZ 85044	rCh0P0uRkcjcChXqVelUaw	33.331821	-111.983
2	113176	113176	False	Active Life; Hotels & Travel; Golf; Event Plan...	Litchfield Park	300 E Wigwam Blvd\nLitchfield Park, AZ 85340	N82S_d9LfAVKi3OS59192A	33.495287	-112.355
3	155492	155492	False	Buffets; Chinese; Restaurants	Phoenix	4909 E Chandler Blvd\nPhoenix, AZ 85048	-Ogv7rpcgUHkFaSy3vD8Sw	33.304138	-111.978
4	40440	40440	False	Hotels & Travel; Airports	Phoenix	3400 E Sky Harbor Blvd\nPhoenix, AZ 85034	hW0Ne_HTHEAgGF1rAdmR-g	33.434750	-112.006

Figure 5: Data Resampling

The Figure 6, checking for missing data sum count for each column.

```
data.isnull().sum()   # checking fo
```

```
index                         0
Unnamed: 0                    0
business_blank                0
business_categories         351
business_city                 0
business_full_address         0
business_id                   0
business_latitude             0
business_longitude            0
business_name                 0
business_neighborhoods   100000
business_open                 0
business_review_count         0
business_stars                0
business_state                0
business_type                 0
cool                          0
date                          0
funny                         0
review_id                     0
reviewer_average_stars        0
reviewer_blank                0
reviewer_cool                 0
reviewer_funny                0
reviewer_name              6176
reviewer_review_count         0
reviewer_type                 0
reviewer_useful               0
stars                         0
text                          3
type                          0
useful                        0
user_id                       0
dtype: int64
```

Figure 6: Class Count

Figure 7 includes a code for analyzing the features and what all values they store.

```
data['business_name'].unique() , len(data['business_name'].unique() )  #checking values for column business neighborhood
```

```
(array(['zpizza', "Ahwatukee Farmers' Market", 'The Wigwam', ...,
        "Earl's Mexican-American Food", 'The Corporate Cafe',
        'Studio C Hair Salon'], dtype=object),
 7789)
```

```
data['business_neighborhoods'].value_counts()   #checking values for column business neighborhood
```

```
Series([], Name: business_neighborhoods, dtype: int64)
```

```
data['business_categories'].value_counts()   #checking values for column business categories
```

```
Mexican; Restaurants                                                        6608
American (New); Restaurants                                                 3463
Pizza; Restaurants                                                          2898
American (Traditional); Restaurants                                         2144
Food; Coffee & Tea                                                          1930
                                                                            ... 
Active Life; Mountain Biking; Hiking; Parks                                    1
Party Supplies; Flowers & Gifts; Shopping; Event Planning & Services; Local Flavor       1
Hotels & Travel; Airport Shuttles; Limos; Transportation; Public Transportation          1
Plumbing; Home Services; Contractors                                           1
Fashion; Shopping; Beauty & Spas; Maternity Wear; Baby Gear & Furniture; Day Spas       1
Name: business_categories, Length: 1988, dtype: int64
```

```
data['business_stars'].value_counts() #checking values for column business stars
```

```
4.0    39416
3.5    26890
4.5    15106
3.0    10365
2.5     3610
5.0     2934
2.0     1179
1.5      319
1.0      181
Name: business_stars, dtype: int64
```

```
data['stars'].value_counts() #checking values for column stars
```

```
4    34831
5    33116
3    15470
2     9022
1     7561
Name: stars, dtype: int64
```

```
data=data.drop(['business_neighborhoods','reviewer_name','Unnamed: 0', 'index'],axis=1) #dropping columns which are same content
```

Figure 7: Column Values

The Figure 8 code segment to check for missing values in each column and treament.

```
data.isnull().sum()
```

```
business_blank              0
business_categories       351
business_city               0
business_full_address       0
business_id                 0
business_latitude           0
business_longitude          0
business_name               0
business_open               0
business_review_count       0
business_stars              0
business_state              0
business_type               0
cool                        0
date                        0
funny                       0
review_id                   0
reviewer_average_stars      0
reviewer_blank              0
reviewer_cool               0
reviewer_funny              0
reviewer_review_count       0
reviewer_type               0
reviewer_useful             0
stars                       0
text                        3
type                        0
useful                      0
user_id                     0
dtype: int64
```

```
data = data.dropna()     #dropping null values from the data
```

Figure 8: Missing Values

As seen in Figure 9, the count of words, upper case and lower case words to analyse the text reviews.

```
#checking the count for each word, uppercase characters and special characters
data['WordCount'] = [len(title.split()) for title in data['text']]
data['UppercaseCount'] = [sum(char.isupper() for char in title) for title in data['text']]
data['SpecialCount'] = [sum(char in string.punctuation for char in title) for title in data['text']]
```

```
data.head(10)   #printing top 10 rows for the data
```

	business_blank	business_categories	business_city	business_full_address	business_id	business_latitude	business_longitude	business_name
0	False	Gluten-Free; Pizza; Vegan; Restaurants	Phoenix	53 West Thomas Road\nPhoenix, AZ 85013	Shl6PtJERnowSJSC4lHbYQ	33.479992	-112.076718	zpizza
1	False	Food; Farmers Market	Phoenix	4700 E Warner Rd\nPhoenix, AZ 85044	rCh0P0uRkcjcChXqVelUaw	33.331821	-111.983203	Ahwatukee Farmers' Marke
2	False	Active Life; Hotels & Travel; Golf; Event Plan...	Litchfield Park	300 E Wigwam Blvd\nLitchfield Park, AZ 85340	N82S_d9LfAVKi3OS59192A	33.495287	-112.355617	The Wigwam
3	False	Buffets; Chinese; Restaurants	Phoenix	4909 E Chandler Blvd\nPhoenix, AZ 85048	-Ogv7rpcgUHkFaSy3vD8Sw	33.304138	-111.978392	Hong Kong Gourmet Buffe
4	False	Hotels & Travel; Airports	Phoenix	3400 E Sky Harbor Blvd\nPhoenix, AZ 85034	hW0Ne_HTHEAgGF1rAdmR-g	33.434750	-112.006440	Phoenix Sky Harbor Internationa Airpor

Figure 9: Word count

5 Sentiment Analysis

The Figure 10, illustrate the sentiment analysis of review and combining the score of user rating and sentiment of review to get the final sentiment.

```
sa = SentimentIntensityAnalyzer()   # initializing sentiment intenstity
```

```
score = lambda title: sa.polarity_scores(title)['compound']     # check
data['scores'] = data['text'].apply(score)
```

```
# Adding user review and restuarent review star to get final sentiment
data['sentiment'] = data['scores'] +  data['stars']
data['sentiment']
```

```
0        4.8625
1        4.9679
2        3.1370
3        4.9784
4        4.9759
          ...
99995    3.2732
99996    4.9559
99997    3.2554
99998    5.9752
99999    3.9058
Name: sentiment, Length: 99646, dtype: float64
```

Figure 10: Sentiment Analysis

The Figure 11, illustrate the code to analyse value for positive or negative class based on final sentiment.

```
s=[]
for compound in data['sentiment']:
        if compound >3:
            s.append(1)   # putting 1 if compour
        else:
            s.append(0)   # putting 0 if compour
print(s)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1
1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
data['sentiment'] = s
```

```
data['sentiment'].value_counts()
```

```
1    81528
0    18118
Name: sentiment, dtype: int64
```

Figure 11: Sentiment Analysis

6 Data Cleaning

The Figure 12, illustrate the function to clear punctuations and contractions of the words in the review.

```
def cleanpunc(sentence): #function to clean the wor
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

```
def decontracted(phrase):
    # This function decontract words like it's to i

    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    phrase = re.sub(r"\n", " ", phrase)
    return phrase
```

Figure 12: Clean punctuations and contractions

The Figure 13, illustrate the stopwords.

```
stop = stopwords.words('english') #set of stopwords
more_stopWords =['well', 'even', 'know', 'one']
stop.extend(more_stopWords)
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't", 'well', 'even', 'know', 'one']
```

Figure 13: Stopwords

The Figure 14, illustrate the code to generate one single review text by combining review text, business category, city and name.

```
data['text'] = data['text'] + ' ' + data['business_categories'] + ' ' + data['business_city'] + ' ' + data['business_name']
```

```
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
data['text']
```

```
0        We tend order Napoli add pepperoni. My ideal p...
1        Love: Dr Hummus--best pita tzatziki since Gree...
2        Absolutely First Class, But EVERYTHING extra, ...
3        I can't say I'm big fan buffets. I never eat e...
4        first observation airport massive. massively o...
                                ...
99995    Pretty standard bar food drink. But thing bet ...
99996    I sleep, I Yelp searching day... Donuts.....I ...
99997    Sit bar, order tacos sangria. disappointed.* *...
99998    The pizza excellent! It's super thin crunchy, ...
99999    Okay maybe second night jitters? Perhaps unfai...
Name: text, Length: 99646, dtype: object
```

Figure 14: Final review Text

The Figure 15, illustrate basic variable initialized for text cleaning.

```
#Basic variables
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
```

Figure 15: Variables

The Figure 16, illustrate the process to initialize stemmer and loop for review text cleaning process checking each word in the review.

```
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemm
sno
```

```
<nltk.stem.snowball.SnowballStemmer at 0x2004bbe4410>
```

```
for sent in tqdm(data['text']):
    filtered_sentence=[]
    sent = decontracted(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (data['sentiment'].values)[i] == 1:
                            all_positive_words.append(s) #list of all words
                    if (data['sentiment'].values)[i] == 0:
                            all_negative_words.append(s) #list of all words
                else:
                    continue
            else:
                continue

    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    final_string.append(str1)
    i+=1

data["clean_review"] = final_string
data['clean_review']=data['clean_review'].str.decode("utf-8")
```

Figure 16: Generating stemmer and Cleaning Process

The Figure 17, illustrate the code to print clean and unclean review.

```
print("Unclean review:",data['text'],"\n")
print("*"*60)
print("\nClean review:",data["clean_review"])
```

```
Unclean review: 0        We tend order Napoli add pepperoni. My ideal p...
1        Love: Dr Hummus--best pita tzatziki since Gree...
2        Absolutely First Class, But EVERYTHING extra, ...
3        I can't say I'm big fan buffets. I never eat e...
4        first observation airport massive. massively o...
                              ...
99995    Pretty standard bar food drink. But thing bet ...
99996    I sleep, I Yelp searching day... Donuts.....I ...
99997    Sit bar, order tacos sangria. disappointed.* *...
99998    The pizza excellent! It's super thin crunchy, ...
99999    Okay maybe second night jitters? Perhaps unfai...
Name: text, Length: 99646, dtype: object

************************************************************

Clean review: 0        tend order napoli add pepperoni ideal pizza al...
1        pita tzatziki sinc greec past summer sell stuf...
2        absolut first class everyth extra wire interne...
3        say big fan buffet never eat enough make worth...
4        first observ airport massiv massiv organ north...
                              ...
99995    pretti standard bar food drink thing bet hors ...
99996    sleep yelp search day donut could back anyon a...
99997    sit bar order taco sangria disappoint might di...
99998    pizza excel super thin crunchi great sauc real...
99999    okay mayb second night jitter perhap unfair re...
Name: clean_review, Length: 99646, dtype: object
```

Figure 17: Clean and unclean review

The Figure 18, illustrate checking for positive words.

```
#count for positive words
pd.DataFrame(pd.Series(all_positive_words).value_counts())
```

	0
b'restaur'	73886
b'place'	60601
b'good'	57846
b'food'	51077
b'great'	46765
...	...
b'vancouverit'	1
b'lookey'	1
b'mycheesesteak'	1
b'carmelo'	1
b'epi'	1

46588 rows × 1 columns

Figure 18: Positive words

The Figure 19, illustrate wordcloud for positive words.

```
#generating word cloud for all postive words
print("Wordcloud of words present in positive class : \n")
wordcloud = WordCloud(width = 500, height = 500,background_color ='white', min_font_size = 10).generate(b'
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

Figure 19: Wordcloud for Positive words

The Figure 20, illustrate checking for negative words.

```
#count for negative words
pd.DataFrame(pd.Series(all_negative_words).value_counts())
```

	0
b'restaur'	17561
b'food'	14483
b'place'	13781
b'like'	12023
b'would'	10671
...	...
b'piedmont'	1
b'obsolesc'	1
b'witic'	1
b'bruckheim'	1
b'wkend'	1

24698 rows × 1 columns

Figure 20: Negative words

The Figure 21, illustrate wordcloud for negative words.

```
#generating word cloud for all negative words
print("Wordcloud of words present in negative class : \n")
wordcloud = WordCloud(width = 500, height = 500,background_color ='white', min_font_size = 10).generate(b'
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

```
Wordcloud of words present in negative class :
```

Figure 21: Wordcloud for Negative words

Figures 22 show the code to split data into training and test set.

```
# Spliting data in test and train data so the models can be trained and tested
n = int(len(data)- (len(data)*0.1))
train= data[0:n]
test = data[n:]
train.shape, train.shape
```

```
((89691, 35), (89691, 35))
```

```
# Separting the features and target data in test and train data so the models can be trained and tested
X_train=train['clean_review']
X_test=test['clean_review']
y_train=train['sentiment']
y_test=test['sentiment']
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((89691,), (9966,), (89691,), (9966,))
```

Figure 22: Train test Split

7 Tokenisation

The Figure 23, illustrate the code for TF-IDF Vectorizer

.

```
#tokenizesing the sentences
vectorizer = TfidfVectorizer(analyzer='word', stop_words='english', max_features=45, lowercase=True,
                             token_pattern='[a-zA-Z0-9]{3,}')
```

```
X_train = vectorizer.fit_transform(X_train).toarray()
```

```
X_test = vectorizer.fit_transform(X_test).toarray()
```

```
X_train.shape, X_test.shape
```

```
((89691, 45), (9966, 45))
```

```
result = pd.DataFrame()
```

Figure 23: Tf-idf Vectorizer

The Figure 24, illustrate the code for Bert Tokenizer.

```
# Spliting data in test and train data set
n = int(len(data) - (len(data)*0.1))
train= data[0:n]
test = data[n:]
train.shape, test.shape
```

```
((89681, 35), (9965, 35))
```

```
X_train=train['clean_review']
X_test=test['clean_review']
y_train=train['sentiment']
y_test=test['sentiment']
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((89681,), (9965,), (89681,), (9965,))
```

```
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased', max_length=956)
```

```
tokenized_texts = [tokenizer.tokenize(com) for com in X_train]
tokenized_texts = [sent[:len(tokenized_texts)] for sent in tokenized_texts]
X_train = [tokenizer.convert_tokens_to_ids(com) for com in tokenized_texts]
X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train, maxlen=50, truncating='post', padding='post')
X_train.shape
```

```
(89681, 50)
```

```
tokenized_texts = [tokenizer.tokenize(com) for com in X_test]
tokenized_texts = [sent[:len(tokenized_texts)] for sent in tokenized_texts]
X_test = [tokenizer.convert_tokens_to_ids(com) for com in tokenized_texts]
X_test = tf.keras.preprocessing.sequence.pad_sequences(X_test, maxlen=50, truncating='post', padding='post')
X_test.shape
```

```
(9965, 50)
```

Figure 24: Bert Tokenizer

8 Machine Learning Models

8.1 DBScan TF-IDF

```
DBSCAN_model = DBSCAN(eps=1, min_samples=25, algorithm='brute', metric='euclidean')
```

```
DBSCAN_model.fit(X_train, y_train)
```

```
DBSCAN(algorithm='brute', eps=1, min_samples=25)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, y_pred)*100
acc
```

```
17.92093116596428
```

```
#Checking the accuracy
f1 = f1_score(y_test, y_pred, average='micro')*100
f1
```

```
17.920931165964284
```

```
#Checking the precision score
prec = precision_score(y_test, y_pred, average='micro')*100
prec
```

```
17.92093116596428
```

```
#Checking the recall score
recal = recall_score(y_test, y_pred, average='weighted')*100
recal
```

```
17.92093116596428
```

```
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix
[[1786    0]
 [8180    0]]

<Axes: >
```

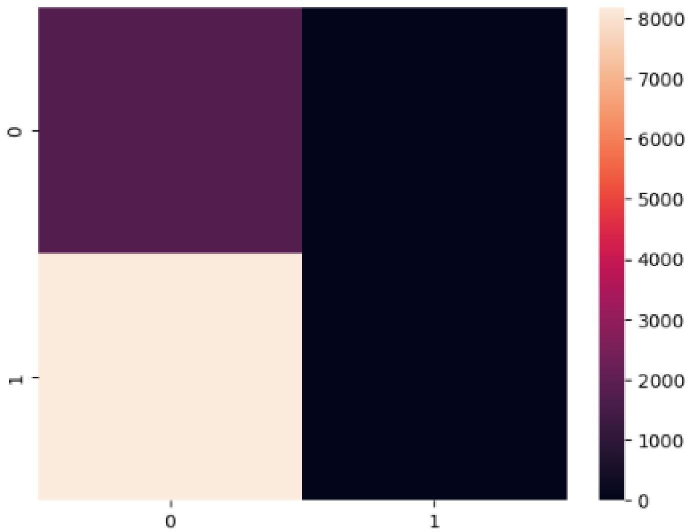


Figure 25: Implementation of DBScan TF-IDF

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.18      1.00      0.30      1786
           1       0.00      0.00      0.00      8180

    accuracy                           0.18      9966
   macro avg       0.09      0.50      0.15      9966
weighted avg       0.03      0.18      0.05      9966
```

Figure 26: Implementation of DBScan TF-IDF

8.2 KMeans TF-IDF

```
KMeans_model = KMeans(n_clusters=2, random_state=0, n_init=5, algorithm='auto')
KMeans_model.fit(X_train, y_train)
```

```
KMeans
KMeans(algorithm='auto', n_clusters=2, n_init=5, random_state=0)
```

```
ypred = KMeans_model.predict(X_test)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, ypred)*100
acc
```

```
45.38430664258479
```

```
#Checking the f1 score
f1 = f1_score(y_test, ypred, average='weighted')*100
f1
```

```
50.5717883923421
```

```
#Checking the precision score
prec = precision_score(y_test, ypred, average='weighted')*100
prec
```

```
73.52387995494142
```

```
# Checking the recall score
recal = recall_score(y_test, ypred, average='weighted')*100
recal
```

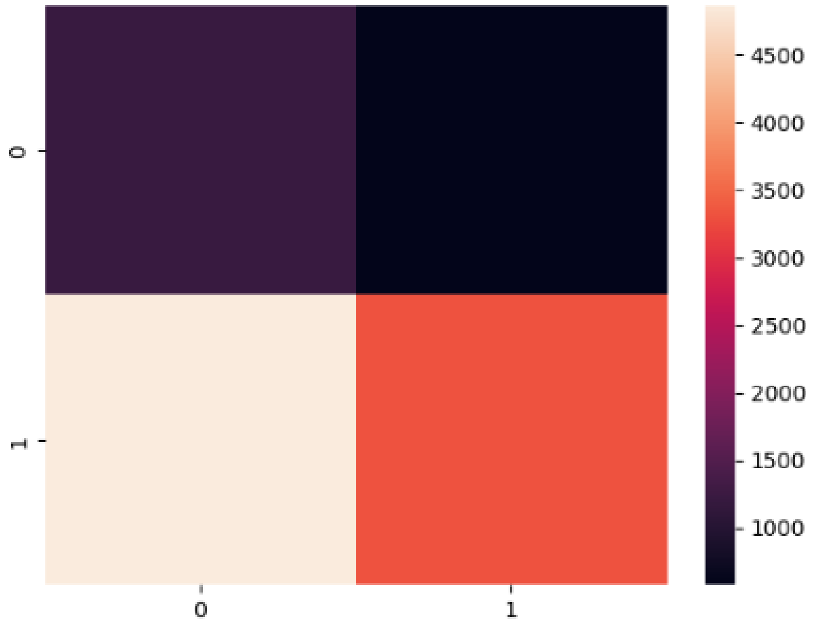
```
45.38430664258479
```

Figure 27: Implementation of KMeans TF-IDF

```
print("Confusion Matrix")
print(confusion_matrix(y_test, ypred))
sns.heatmap(confusion_matrix(y_test, ypred))
```

```
Confusion Matrix
[[1212  574]
 [4869 3311]]
```

```
<Axes: >
```



Precision measures how many of the instances predicted as positive are actually positive, wh
correctly predicted as positive.

In the context of K-means clustering, precision and recall are not typically used as evaluation
It doesn't assign class labels explicitly, and the classes assigned by K-means may not corres

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.20      0.68      0.31      1786
           1       0.85      0.40      0.55      8180

    accuracy                           0.45      9966
   macro avg       0.53      0.54      0.43      9966
weighted avg       0.74      0.45      0.51      9966
```

Figure 28: Implementation of KMeans TF-IDF

8.3 KNN TF-IDF

```
KNC_model = KNeighborsClassifier(algorithm= 'auto', metric= 'cosine', n_neighbors= 35)
KNC_model.fit(X_train, y_train)
```

```
▾            KNeighborsClassifier
KNeighborsClassifier(metric='cosine', n_neighbors=35)
```

```
ypred = KNC_model.predict(X_test)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, ypred)*100
acc
```

```
82.52056993778848
```

```
#Checking the f1 score
f1 = f1_score(y_test, ypred, average='micro')*100
f1
```

```
82.52056993778848
```

```
#Checking the precision score
prec = precision_score(y_test, ypred, average='weighted')*100
prec
```

```
78.65158222904826
```

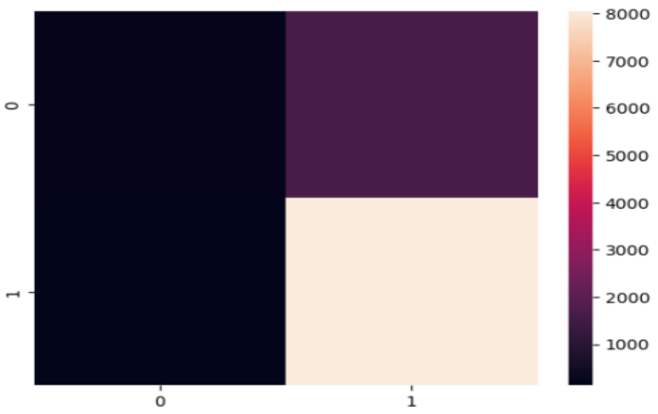
```
# Checking the recall score
recal = recall_score(y_test, ypred, average='weighted')*100
recal
```

```
82.52056993778848
```

Figure 29: Implementation of KNN TF-IDF

```
print("Confusion Matrix")
print(confusion_matrix(y_test, ypred))
sns.heatmap(confusion_matrix(y_test, ypred))
```

```
Confusion Matrix
[[ 171 1615]
 [ 127 8053]]
<Axes: >
```




```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.57      0.10      0.16      1786
           1       0.83      0.98      0.90      8180

    accuracy                           0.83      9966
   macro avg       0.70      0.54      0.53      9966
weighted avg       0.79      0.83      0.77      9966
```

Figure 30: Implementation of KNN TF-IDF

8.4 DBSCan Bert

```
DBS_model = DBSCAN(eps=2, min_samples=5, algorithm='auto', metric='cosine')
KMeans_model.fit(X_train, y_train)
```

```
KMeans
KMeans(algorithm='auto', n_clusters=2, n_init=5, random_state=0)
```

```
#Applying our function
y_pred = DBS_model.fit_predict(X_test, y_test)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, y_pred)*100
acc
```

```
17.92093116596428
```

```
#Checking the accuracy
f1 = f1_score(y_test, y_pred, average='macro')*100
f1
```

```
15.197413206262766
```

```
#Checking the precision score
prec = precision_score(y_test, y_pred, average='macro')*100
prec
```

```
8.96046558298214
```

```
#Checking the recall score
recal = recall_score(y_test, y_pred, average='macro')*100
recal
```

```
50.0
```

Figure 31: Implementation of DBSCan Bert

```
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix
[[1786    0]
 [8180    0]]

<Axes: >
```

The confusion matrix is a useful tool to evaluate the performance of a classification al
ability to correctly identify instances of the positive class. The absence of True Positive
algorithm is not performing well in identifying instances of the positive class. Dependir
need to adjust the algorithm or consider other evaluation metrics to improve its perforr

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.18      1.00      0.30      1786
           1       0.00      0.00      0.00      8180

    accuracy                           0.18      9966
   macro avg       0.09      0.50      0.15      9966
weighted avg       0.03      0.18      0.05      9966
```

Figure 32: Implementation of DBSCan Bert

8.5 KMeans Bert

```
KMeans_model = KMeans(n_clusters=2, random_state=0, n_init=5, algorithm='auto')
KMeans_model.fit(X_train, y_train)
```

```
KMeans
KMeans(algorithm='auto', n_clusters=2, n_init=5, random_state=0)
```

```
ypred = KMeans_model.predict(X_test)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, ypred)*100
acc
```

```
33.03230985350191
```

```
#Checking the f1 score
f1 = f1_score(y_test, ypred, average='weighted')*100
f1
```

```
35.00233108242926
```

```
#Checking the precision score
prec = precision_score(y_test, ypred, average='weighted')*100
prec
```

```
71.40918144109408
```

```
# Checking the recall score
recal = recall_score(y_test, ypred, average='weighted')*100
recal
```

```
33.03230985350191
```

Figure 33: Implementation of KMeans Bert

```
print("Confusion Matrix")
print(confusion_matrix(y_test, ypred))
sns.heatmap(confusion_matrix(y_test, ypred))
```

```
Confusion Matrix
[[1399  387]
 [6287 1893]]

<Axes: >
```

The model seems to have relatively low recall, indicating that it is not effectively captu
goals of your model, you might need to adjust the algorithm or its parameters to impr

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       0.18      0.78      0.30      1786
           1       0.83      0.23      0.36      8180

    accuracy                           0.33      9966
   macro avg       0.51      0.51      0.33      9966
weighted avg       0.71      0.33      0.35      9966
```

Figure 34: Implementation of KMeans Bert

8.6 KNN Bert

```
KNN_model = KNeighborsClassifier(algorithm='brute', metric='cosine', n_neighbors=35, weights='distance')
KNN_model.fit(X_train, y_train)
```

```
KNeighborsClassifier

KNeighborsClassifier(algorithm='brute', metric='cosine', n_neighbors=35,
                     weights='distance')
```

```
ypred = KNN_model.predict(X_test)
```

```
#Checking the accuracy
acc = accuracy_score(y_test, ypred)*100
acc
```

```
82.09913706602448
```

```
#Checking the f1 score
f1 = f1_score(y_test, ypred, average='micro')*100
f1
```

```
82.09913706602448
```

```
#Checking the precision score
prec = precision_score(y_test, ypred, average='micro')*100
prec
```

```
82.09913706602448
```

```
# Checking the recall score
recal = recall_score(y_test, ypred, average='weighted')*100
recal
```

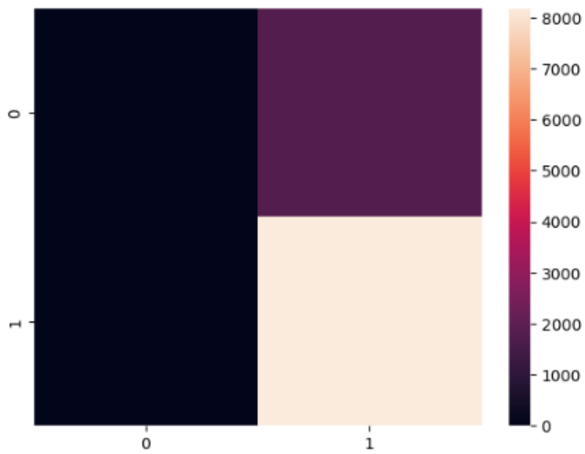
```
82.09913706602448
```

Figure 34: Implementation of KNN Bert

```
print("Confusion Matrix")
print(confusion_matrix(y_test, ypred))
sns.heatmap(confusion_matrix(y_test, ypred))
```

```
Confusion Matrix
[[   2 1784]
 [   0 8180]]
```

```
<Axes: >
```




```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.00      0.00      1786
           1       0.82      1.00      0.90      8180

    accuracy                           0.82      9966
   macro avg       0.91      0.50      0.45      9966
weighted avg       0.85      0.82      0.74      9966
```

Figure 35: Implementation of KNN Bert

8.7 Surprise

```
reader = Reader(rating_scale=(0, 9))
data_surprise = Dataset.load_from_df(data[['scores', 'business_stars', 'stars']], reader)
```

```
benchmark = []
# Iterate over all algorithms

algorithms = [SVD(), KNNBasic(), KNNWithMeans()]

print ("Attempting: ", str(algorithms), '\n\n')

for algorithm in algorithms:
    print("Starting: " ,str(algorithm))
    # Perform cross validation
    results = cross_validate(algorithm, data_surprise, measures=['RMSE'], cv=3, verbose=False)
    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algorithm']))
    benchmark.append(tmp)
    print("Done: " ,str(algorithm), "\n")

print ('\n\tDONE\n')
```

```
Attempting:  [<surprise.prediction_algorithms.matrix_factorization.SVD object at 0x00000200193C2110>, <surprise.prediction_algo
rithms.knns.KNNBasic object at 0x000002001943FDD0>, <surprise.prediction_algorithms.knns.KNNWithMeans object at 0x00000201C8251
FD0>]


Starting:  <surprise.prediction_algorithms.matrix_factorization.SVD object at 0x00000200193C2110>
Done:  <surprise.prediction_algorithms.matrix_factorization.SVD object at 0x00000200193C2110>

Starting:  <surprise.prediction_algorithms.knns.KNNBasic object at 0x000002001943FDD0>
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.
Done:  <surprise.prediction_algorithms.knns.KNNBasic object at 0x000002001943FDD0>

Starting:  <surprise.prediction_algorithms.knns.KNNWithMeans object at 0x00000201C8251FD0>
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.
Done:  <surprise.prediction_algorithms.knns.KNNWithMeans object at 0x00000201C8251FD0>


	DONE
```

Figure 36: Models using Surprise

9 Model result

This section explains the performance of the models.

9.1 Model Scores

```
result.columns = ['Model', 'Accuracy', 'F1-Score', 'Precision', 'Recall']
result
```

	Model	Accuracy	F1-Score	Precision	Recall
0	DBSCan TF-IDF	17.920931	17.920931	17.920931	17.920931
0	KMeans TF-IDF	45.384307	50.571788	73.523880	45.384307
0	KNN TF-IDF	82.520570	82.520570	78.651582	82.520570
0	DBSCan Bert	17.920931	15.197413	8.960466	50.000000
0	KMeans Bert	33.032310	35.002331	71.409181	33.032310
0	KNN Bert	82.099137	82.099137	82.099137	82.099137

Figure 37: Model Performance Traditional

```
surprise_results = pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

```
surprise_results
```

	test_rmse	fit_time	test_time
Algorithm			
KNNBasic	1.118203	204.141678	631.550080
KNNWithMeans	1.118270	195.671062	600.469911
SVD	1.203250	1.089043	1.197574

Figure 38: Model Performance Surprise

```
knn_recomm = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
knn_recomm.fit(X_train)
```

```
NearestNeighbors(algorithm='brute', metric='cosine')
```

In a Jupyter environment, please rerun this cell to show the HTML representation o
On GitHub, the HTML representation is unable to render, please try loading this pag

```
distances, indices = knn_recomm.kneighbors(X_test[0].reshape(1, -1), n_n
print('Recommendations:\n')
for j in indices:
       print(data['business_name'].iloc[j])
```

```
Recommendations:

76811            Arizona Sandwich Company
69113                  Whole Foods Market
24971                       Phoenix Pride
48155                         Havana Café
34731             Lalibela Ethiopian Cafe
74712     Sheraton Phoenix Downtown Hotel
43883                  Seamus McCaffrey's
75240             Two Hippies Beach House
61386       Green New American Vegetarian
29944                 Aunt Chilada's Tempe
71497                   Zipps Sports Grill
Name: business_name, dtype: object
```

Figure 39: Recommendation System

References

https://data.world/brianray/yelp-reviews

https://surpriselib.com/

TfidfVectorizer

BertTokenizer

VADER Sentiment Analyzer

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

https://scikit-learn.org/stable/modules/neighbors.html

https://www.analyticsvidhya.com/blog/2020/08/recommendation-system-k-nearest-neighbors/