

# Data Wrangling: Group 130

**Chaitanya Tambolkar Student ID: 34121315**

**Fahmid Tawsif Khan Chowdhury Student ID: 34093117**

## TASK 5 - Development History

### Task 1

**Date: 05/08/2024 (Monday)**

#### Description:

We began the assignment by going through the data provided in the .xlsx and .txt files. Our focus was to understand the data structure and identify any potential challenges in parsing and preprocessing the data. The exploration was aimed at laying the groundwork for effective data manipulation in the subsequent steps.

#### Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**
  - Explored the .xlsx file to understand the structure and content of each sheet, focusing on identifying any missing values and empty columns.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**
  - Examined the .txt files for XML format issues and noted variations in tag names and capitalization.

#### Proof:

```
3.1. Examine the .xlsx file

[ ] xlsx_file_path = "/Users/fahmidawsifkhanchowdhury/Documents/Monash/53/FIT5196/Assignment 1/student_group130/group130.xlsx"

[ ] # Read the xlsx file
data_xlsx = pd.read_excel(f"{xlsx_file_path}",
                          sheet_name=['Sheet0', 'Sheet1', 'Sheet2', 'Sheet3', 'Sheet4', 'Sheet5', 'Sheet6', 'Sheet7', 'Sheet8', 'Sheet9', 'Sheet10', 'Sheet11', 'Sheet12', 'Sheet13', 'Sheet14',
                          # Check the sheet names
                          data_xlsx.keys()

dict_keys(['Sheet0', 'Sheet1', 'Sheet2', 'Sheet3', 'Sheet4', 'Sheet5', 'Sheet6', 'Sheet7', 'Sheet8', 'Sheet9', 'Sheet10', 'Sheet11', 'Sheet12', 'Sheet13', 'Sheet14', 'Sheet15'])

[ ] # Check Sheet 3
data_xlsx["Sheet3"].head()
```

	x4	x3	x2	x1	user_id	name	time	rating	text	pics	resp	gmap_id
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	107863153433258785850	Tim Tran	1.523534e+12	5.0	I have gone to Dr LU several times for root ca...	NaN	NaN	0x80dd27372ac1b4b9:0x763012fc9a6a0485
4	NaN	NaN	NaN	NaN	107137468803053365925	Andrea Thornton	1.610398e+12	5.0	Excellent service.	NaN	NaN	0x80dd27372ac1b4b9:0x763012fc9a6a0485

### 3.2. Examine the .txt files

```
[ ] # Load all txt file and read them

txt_files_path = "/Users/fahmidtawsifkhanchowdhury/Documents/Monash/S3/FIT5196/Assignment 1/student_group130/"

file_paths = [
    f"{txt_files_path}group130_0.txt",
    f"{txt_files_path}group130_1.txt",
    f"{txt_files_path}group130_2.txt",
    f"{txt_files_path}group130_3.txt",
    f"{txt_files_path}group130_4.txt",
    f"{txt_files_path}group130_5.txt",
    f"{txt_files_path}group130_6.txt",
    f"{txt_files_path}group130_7.txt",
    f"{txt_files_path}group130_8.txt",
    f"{txt_files_path}group130_9.txt",
    f"{txt_files_path}group130_10.txt",
    f"{txt_files_path}group130_11.txt",
    f"{txt_files_path}group130_12.txt",
    f"{txt_files_path}group130_13.txt",
    f"{txt_files_path}group130_14.txt"
]

all_files = []

for i in file_paths:
    with open(i, 'r', encoding="utf-8") as file:
        content = file.read()
        all_files.append(content)

text = ''.join(all_files)
```

---

**Date: 07/08/2024 (Wednesday)**

## Task 2

### Description:

The data loading and initial parsing process began today. We focused on reading all relevant sheets from the .xlsx file into pandas DataFrames and examining the structure of the text files to identify patterns that could be used for parsing the semi-structured data.

### Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**
  - Loaded the .xlsx sheets into pandas and explored the key columns required for further analysis.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**

- Created initial regular expressions to parse the text files and extract key information, such as user ID, review text, and timestamps.

## Proof:

### 4.1. Parsing and Preprocessing - .xlsx

In this section, the files are parsed and processed to prepare the dataset for further analysis. The relevant columns (user\_id, name, time, rating, text, pics, resp, gmap\_id) are extracted from each sheet in the .xlsx. These sheets are then concatenated into a single DataFrame, combining all the review data into one structure. The DataFrame is examined for missing values and data types. Empty strings are replaced with NaN values to standardise the missing data representation. The time column, containing Unix timestamps, is converted from float to integer and the missing timestamps are set to NaN. This preprocessing step ensures that the data is clean and structured.

```
[ ] # Extract the column names from the excel file
listofsheets = []

for i in data.xlsx.values():
    df = i[["user_id", "name", "time", "rating", "text", "pics", "resp", "gmap_id"]]
    listofsheets.append(df)

print(len(listofsheets))
```

16

```
[ ] # Concatenate the sheets into one dataframe
df_concat = pd.concat(listofsheets)
df_concat
```

### 4.2. Parsing and Preprocessing - .txt

In this section, we perform a series of string replacements to clean up and format the content. The operations include removing unnecessary spaces after angle brackets, replacing double slashes with single slashes, removing newline characters, and eliminating specific XML tags such as <dataset> and </dataset>. Additionally, occurrences of the word "None" are removed, and the XML version declaration (?xml version="1.0" encoding="UTF-8"?>) is stripped out.

```
[ ] # Perform a series of text replacements to clean up the content
text_replace = text.replace("< ", "<")
text_replace = text_replace.replace("//", "/")
text_replace = text_replace.replace("\n", "")
text_replace = text_replace.replace("<dataset>", "")
text_replace = text_replace.replace("</dataset>", "")
text_replace = text_replace.replace("None", "")

# Remove the XML version declaration
text_replace = text_replace.replace(""?xml version="1.0" encoding="UTF-8"?>", "")

# text_replace
```

```
[ ] # Check total number of records
print(text_replace.count("<record>"))
print(text_replace.count("</record>"))
```

34528  
34528

```
[ ] pattern = r'<([>]+)>(.*?)<\/\1>'

records = re.findall(r'<record>(.*?)<\/record>', text_replace)

data = []

for r in records:
    all_records = dict(re.findall(pattern, r))
    data.append(all_records)

df_text = pd.DataFrame(data)
```

```
[ ] df_text.head()
```

	user	user_name	Time	rate	review	pics	Resp	Gmap_id	name	date	...	gmap_id	
0	106123641366344298666	Turbo Slug	1618115398252	5	Loved the beer and the scenery, I'm definitely...	[[{"url": "https://lh5.googleusercontent.com/p/...",	0x80c27e755ec02345:0x5d476bb907e17e4e		NaN	NaN	...	NaN	1
1	101668037073551058819	NaN	NaN	NaN	Great beer, always has a food truck but live m...		NaN	NaN	Sara Jones	1620440337570	...	NaN	
2	NaN	Doris Alexander	NaN	NaN	NaN		NaN	NaN	NaN	1612955398876	...	NaN	
3	NaN	Melinda Miramon	NaN	5	NaN	NaN	NaN	NaN	NaN	1561411093247	...	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	...	NaN	

name	date	...	gmap_id	Text	pictures	rating	Pictures	Response	username	userid	Date	Rate
NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Sara Jones	1620440337570	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	1612955398876	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	1561411093247	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

### Task 3

Date: 12/08/2024 (Monday)

#### Description:

Today, we moved forward with data preprocessing, focusing on merging and cleaning the datasets. This included removing XML special characters, handling missing data, and merging columns with similar information from different sheets. Additionally, we started converting Unix timestamps to human-readable dates.

#### Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**
  - Focused on cleaning the data by replacing empty strings with NaN and removing special characters.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**
  - Merged similar columns across the DataFrames and converted the Unix timestamps to datetime format.

#### Proof:

- Screenshots showing the code for data cleaning, column merging, and timestamp conversion.



```
# Arrange the columns as per the excel file
df_text = df_text.reindex(columns=['user_id_merged', 'name_merged', 'time_merged', 'rating_merged',
                                   'text_merged', 'pics_merged', 'resp_merged', 'gmap_id_merged'])
df_text.head()
```

	user_id_merged	name_merged	time_merged	rating_merged	text_merged	pics_merged	resp_merged	gmap_id_merged
0	106123641366344298666	Turbo Slug	1618115398252	5	Loved the beer and the scenery, I'm definitely...	['url': 'https://lh5.googleusercontent.com/p/...	0x80c27e755ec02345:0x5d476bb907e17e4e	
1	101668037073551058819	Sara Jones	1620440337570	4	Great beer, always has a food truck but live m...		0x80c27e755ec02345:0x5d476bb907e17e4e	
2	118231381474176364386	Doris Alexander	1612955398876	4	Ate this for lunch and let me tell you, it is ...		0x80c27e755ec02345:0x5d476bb907e17e4e	
3	113424580915165120195	Melinda Miramon	1561411093247	5	My family threw me a retirement party at the B...	['url': 'https://lh5.googleusercontent.com/p/...	0x80c27e755ec02345:0x5d476bb907e17e4e	
4	112288176908265223415	Lynn Brand	1609900450190	5	great food awesome service and always super cle...		0x80c27e755ec02345:0x5d476bb907e17e4e	

```
[ ] # Check for Columns with Only Whitespace Values

columns_to_clean = [
    'user_id_merged', 'name_merged', 'time_merged', 'rating_merged',
    'text_merged', 'pics_merged', 'resp_merged', 'gmap_id_merged'
]

# Regex to check strings with only whitespaces
whitespace_counts = df_text[columns_to_clean].apply(lambda col: col.str.contains(r'^\s*$', na=False).sum())

print("Whitespace-only count:")
print(whitespace_counts)
```

```
⇒ Whitespace-only count:
user_id_merged    0
name_merged       0
time_merged       0
rating_merged     0
text_merged       0
pics_merged       0
resp_merged       0
gmap_id_merged    0
dtype: int64
```

```
[ ] # Check for any duplications
df.duplicated().sum()
```

```
⇒ 873
```

```
[ ] # Drop duplicates
df.drop_duplicates(inplace=True)
```

```
[ ] # Check duplicates
df.duplicated().sum()
```

```
⇒ 0
```

```
[ ] # Reset the index
df.reset_index(drop=True, inplace=True)
```

```
[ ] # Check nulls
df.isnull().sum()
```

```
⇒ user_id      1
   name        1
   time        1
   rating      1
   text    15389
   pics    34434
   resp    31389
   gmap_id     1
dtype: int64
```

```
[ ] # Convert unix timestamp to datetime
df['time'] = pd.to_datetime(df['time'], unit='ms', utc=True)
df['time'] = pd.to_datetime(df['time']).dt.strftime('%Y-%m-%d %H:%M:%S')
df.head(1)
```

	user_id	name	time	rating	text	pics	resp	gmap_id
0	106123641366344298666	Turbo Slug	2021-04-11 04:29:58	5.0	loved the beer and the scenery, i'm definitely...	[[{'url': 'https://lh5.googleusercontent.com/p/...'}]]	<NA>	0x80c27e755ec02345:0x5d476bb907e17e4e

```
# Create if_pic column to indicate presence or absence of data in pics
df['if_pic'] = df['pics'].apply(lambda x: 'Y' if pd.notna(x) and len(str(x)) > 0 else 'N')
df.head(1)
```

	user_id	name	time	rating	text	pics	resp	gmap_id	if_pic
0	106123641366344298666	Turbo Slug	2021-04-11 04:29:58	5.0	loved the beer and the scenery, i'm definitely...	[[{'url': 'https://lh5.googleusercontent.com/p/...'}]]	<NA>	0x80c27e755ec02345:0x5d476bb907e17e4e	Y

```
# Replace NaN to None in text column
df['text'] = df['text'].apply(lambda x: 'None' if pd.isna(x) else x)
df.tail()
```

	user_id	name	time	rating	text	pics	resp	gmap_id	if_pic
35503	117046212642192439198	Gorealla One	2019-05-05 16:15:06	5.0	None	NaN	NaN	0x80d954ac1e876e91:0xff78dd4fb28b4298	N
35504	101941170861041101000	Logan M	2020-04-09 18:08:48	5.0	None	NaN	NaN	0x80d954ac1e876e91:0xff78dd4fb28b4298	N

Task 4:

Date: 14/08/2024 (Wednesday)

Description:

We continued the data preprocessing today, focusing on further consolidating columns and handling missing values. We also began parsing and cleaning the text data, including removing non-English reviews and emojis, ensuring the text was standardized and ready for analysis.

Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**
  - Removed non-English reviews and standardized the text data by converting it to lowercase.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**
  - Developed regex patterns to clean the text data by removing emojis and unwanted characters.

## Proof:

```
[ ] # Convert text column to lower case
df['text'] = df['text'].str.lower()

# Replace (translated by google) with ""
df['text'] = df['text'].str.replace(r'\(translated by google\) ', '', regex=True)

# Remove all languages other than english by using regex r'\(original\).*'
df['text'] = df['text'].str.replace(r'\(original\).*', '', regex=True)

[ ] print(df.iloc[289, 4])
print()
print(df.iloc[737, 4])
print()
print(df.iloc[738, 4])
```

the food is infective! we ordered a burger, which was ok and an "open faced turkey" a typical american dish.the turkey served is not turkey may be a reconstituted turkey served with a fields of baseball, basketball, cricket, volleyball, golf, well-equipped soccer. they are located in the center of fremont in california in the central park area. also open in the ever good air quality

```
[ ] emoji_pattern = re.compile("[
    u\"\\U0001F600-\\U0001F64F\"
    u\"\\U0001F300-\\U0001F5FF\"
    u\"\\U0001F680-\\U0001F6FF\"
    u\"\\U0001F1E0-\\U0001F1FF\"
    u\"\\U00002702-\\U000027B0\"
    u\"\\U000024C2-\\U0001F251\""]+", re.UNICODE)
```

```
# Sample a review with emoji
df.iloc[18, 4]
```

'wolf creek is absolutely awesome! ❤️ great people, awesome food and my new favorite beer 🍷👍 what a treat❤️😋thank babe👶❤️'

```
[ ] df['text'] = df['text'].apply(lambda x: emoji_pattern.sub(r'', x) if isinstance(x, str) else x)
```

```
[ ] df.iloc[18, 4]
```

'wolf creek is absolutely awesome! great people, awesome food and my new favorite beer what a treatthank babe'

In this step, we removed any emojis. We defined a regular expression pattern (emoji\_pattern) that matches a wide range of emojis, including facial expressions, objects, symbols, and country flags. Using this pattern, we applied a lambda function to the text column, which substitutes any matched emojis with an empty string, effectively removing them from the text.

---

## Task 5:

**Date: 19/08/2024 (Monday)**

### Description:

Today, we focused on data aggregation and final cleanup. We grouped the data by gmap\_id to calculate key metrics such as the number of reviews, the number of text entries, and the response count. Additionally, we ensured all remaining inconsistencies, such as whitespace-only entries, were addressed.

### Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**



- Grouped the data and calculated the review counts, text counts, and response counts.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**
  - Handled the final cleanup, ensuring no columns contained only whitespace, and all data types were correct.

**Proof:**

```
# Creating the dataframe for the csv file by grouping the gmap_id and considering other columns
df_csv_output = df.copy()

df_csv_output['text'] = df_csv_output['text'].apply(lambda x: pd.NA if x == 'None' else x)

df_grouped = df_csv_output.groupby('gmap_id').agg(review_count=("rating", "count"),
                                                  review_text_count=("text", "count"),
                                                  response_count=("resp", "count")).reset_index()

df_grouped.iloc[0:20, : ]
```

	gmap_id	review_count	review_text_count	response_count
0	0x54d29333b3ff2583:0xa4c448dcc281f718	87	46	8
1	0x8080354faaaaaaab:0x12fa86bfac99deaf	358	223	0
2	0x808158dba147f905:0x4bf9cfb6c3e27d6b	166	108	0
3	0x8082d8e8700d7197:0x8289cde69284de06	68	36	0
4	0x808418adf88d73d7:0x9f193f3ea32a50	198	120	0
5	0x8084388e140e3dbd:0x45b28a2f335dedb9	158	104	77
6	0x808447ff6954c14f:0x6223414a002d65ab	138	59	0
7	0x8085141569c29f3d:0x9d02d1e9587ba34c	294	229	176
8	0x80853d65eab67323:0xdaaebf75226ebac4	54	40	32
9	0x8085793210ba25d3:0x7e520d94f0750e9b	68	38	0
10	0x808580886dd516f5:0x68b81b89b851262a	523	236	0
11	0x80858094dab29b35:0x18e53a5cdbb55172	175	160	17
12	0x808580bed8e90009:0x4bfcacfeba03db0c	631	347	2

```

# Creating the json file by grouping on the gmap_id and assigning each of the appropriate variables
json_data = {}

json_group = df.groupby('gmap_id')

for gmap_id, group in json_group:
    all_data = {
        "reviews": [],
        "earliest_review_date": group['first_review_date'].iloc[0],
        "latest_review_date": group['last_review_date'].iloc[0]
    }

    for _, row in group.iterrows():
        review = {
            "user_id": row['user_id'],
            "time": row['time'],
            "review_rating": row['rating'],
            "review_text": row['text'],
            "if_pic": row['if_pic'],
            "pic_dim": row['pic_dim'],
            "if_response": row['if_response']
        }
        all_data["reviews"].append(review)

    json_data[gmap_id] = all_data

[ ] first_key = list(json_data.keys())[0]
first_value = json_data[first_key]

print(f"First key: {first_key}")
print(f"First value: {first_value}")

```

## Task 6

**Date: 21/08/2024 (Wednesday)**

### Description:

The final step involved exporting the cleaned and processed data into the required formats (CSV and JSON). We structured the JSON file by grouping reviews under each gmap\_id and included additional metadata such as the earliest and latest review dates.

### Contributions:

- **Team member 1: Chaitanya Sanjay Tambolkar**
  - Structured and exported the JSON file, ensuring it met the required format.
- **Team member 2: Fahmid Tawsif Khan Chowdhury**
  - Exported the CSV file with the aggregated data, ensuring all metrics were correctly calculated and included.

## Proof:

```
# Creating the json file by grouping on the gmap_id and assigning each of the appropriate variables
json_data = {}

json_group = df.groupby('gmap_id')

for gmap_id, group in json_group:
    all_data = {
        "reviews": [],
        "earliest_review_date": group['first_review_date'].iloc[0],
        "latest_review_date": group['last_review_date'].iloc[0]
    }

    for _, row in group.iterrows():
        review = {
            "user_id": row['user_id'],
            "time": row['time'],
            "review_rating": row['rating'],
            "review_text": row['text'],
            "if_pic": row['if_pic'],
            "pic_dim": row['pic_dim'],
            "if_response": row['if_response']
        }
        all_data["reviews"].append(review)

    json_data[gmap_id] = all_data
```

```
[ ] # Output csv file
csv_output = f'{destination_path}/task1_130.csv'
df_grouped.to_csv(csv_output, index=False, encoding='utf-8')
```

	A	B	C	D	E
1	gmap_id	review_count	review_text	response_count	
2	0x54d2933	87	46	8	
3	0x8080354	358	223	0	
4	0x808158c	166	108	0	
5	0x8082d8e	68	36	0	
6	0x808418a	198	120	0	
7	0x808438e	158	104	77	
8	0x808447f	138	59	0	
9	0x8085141	294	229	176	
10	0x80853d6	54	40	32	
11	0x8085793	68	38	0	
12	0x808580e	523	236	0	
13	0x808580e	175	160	17	
14	0x808580e	631	347	2	
15	0x808580e	110	87	10	
16	0x808582e	88	51	31	
17	0x8085871	52	41	0	
18	0x8085874	86	56	0	
19	0x808587e	346	196	127	
20	0x8085970	56	19	0	
21	0x80859a6	238	155	0	
22	0x8085b41	167	116	0	
23	0x808de43	196	75	0	
24	0x808e20e	118	72	0	
25	0x808e3fe	293	207	5	
26	0x808e6a9	78	57	0	
27	0x808e6a9	55	23	0	
28	0x808f79d	125	75	83	
29	0x808f7c3	74	39	0	
30	0x808f7c5	234	111	1	
31	0x808f7e0	184	100	139	

## **TASK 2 - Development History**

### **Task 1: Introduction**

**Date: 22/08/2024 (Thursday)**

#### **Description:**

The initial task was to understand the requirements of Task 2 in Assessment 1 for FIT5196. The primary focus was on processing and analyzing textual data extracted from user reviews, requiring extensive cleaning, tokenization, and transformation into structured formats. The goal was to prepare the data for subsequent natural language processing tasks.

#### **Contributions:**

- **Team member 1: Fahmid Tawsif Khan Chowdhury**
  - Drafted the initial outline for the task and identified the necessary preprocessing steps.
- **Team member 2: Chaitanya Tambolkar**
  - Reviewed the task requirements and provided suggestions for the text processing workflow.

#### **Proof:**

We closely followed Week 5 applied for text **pre-processing** for analysing the steps required for Task 2. The notes for those steps are as follows:

#### **Notes on Preprocessing Steps:**

##### **1. Tokenization:**

- Extracting alphabetic tokens while removing irrelevant characters.
- Eliminating short tokens with less than three characters.

##### **2. Stopwords Removal:**

- **Context-Independent Stopwords:** Removing common English stopwords using a predefined list.
- **Context-Dependent Stopwords:** Identifying and removing high-frequency words that appear in more than 95% of the businesses.

##### **3. Rare Tokens Removal:**

- Filtering out tokens that appear in less than 5% of the businesses.

#### 4. Stemming:

- Reducing tokens to their root forms using the Porter Stemming algorithm.

#### 5. Bigram Extraction:

- Extracting meaningful bigrams using the PMI (Pointwise Mutual Information) measure.

---

### Task 2: Importing Libraries

**Date:** 24/08/2024 (Sat)

#### **Description:**

We imported the necessary Python libraries required for text processing. This included libraries like pandas, nltk, sklearn, and multiprocessing for data manipulation, tokenization, stemming, vectorization, and parallel processing.

#### **Contributions:**

- **Team member 1: Fahmid Tawsif Khan Chowdhury**
  - Imported and configured the nltk library for text processing.
- **Team member 2: Chaitanya Tambolkar**
  - Set up the environment and imported pandas, sklearn, and multiprocessing libraries for data manipulation, vectorization, and parallel processing.

#### **Proof:**

```
[ ] import os
import re
import json
import pandas as pd
import multiprocessing
from itertools import chain
import nltk
from nltk.probability import *
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import MWETokenizer
from nltk.stem import PorterStemmer
from nltk.util import ngrams
from sklearn.feature_extraction.text import CountVectorizer
```

### Task 3: Examining Input File

Date: 25/08/2024 (sat)

#### Description:

We examined the input files, including the JSON and CSV files, to understand their structure and content. The JSON file contained review data, while the CSV file had metadata like review counts.

#### Contributions:

- **Team member 1: Fahmid Tawsif Khan Chowdhury**
  - Loaded and parsed the JSON file, examining the structure and content.
- **Team member 2: Chaitanya Tambolkar**
  - Loaded the CSV file using pandas and explored the first few rows to understand the data distribution.

#### Proof:

##### 3. Examining Input File

```
[ ] # from google.colab import drive
    # drive.mount('/content/drive')

[ ] json_file_path = '/Users/fahmidtawsifkhanchowdhury/Documents/Monash/S3/FIT5196/Assignment 1/Submissions/task1_130.json'
    csv_file_path = '/Users/fahmidtawsifkhanchowdhury/Documents/Monash/S3/FIT5196/Assignment 1/Submissions/task1_130.csv'

[ ] with open(json_file_path, 'r', encoding='utf-8') as file:
    json_data = json.load(file)

    df_csv = pd.read_csv(csv_file_path, encoding='utf-8')
```

Let's examine what is the content of the file.

```
[ ] df_csv.head()
```



	gmap_id	review_count	review_text_count	response_count
0	0x54d29333b3ff2583:0xa4c448dcc281f718	87	46	8
1	0x8080354faaaaaaab:0x12fa86bfac99deaf	358	223	0
2	0x808158dba147f905:0x4bf9cfb6c3e27d6b	166	108	0
3	0x8082d8e8700d7197:0x8289cde69284de06	68	36	0
4	0x808418adf88d73d7:0x9f193f3ea32a50	198	120	0

For this purpose, we need to filter the gmap\_ids of the businesses that have review\_count more than 70.

In this step, the dataset is filtered to focus on businesses with more than 70 text reviews. The dataframe df\_csv is filtered to extract gmap\_ids from businesses meeting this criterion, which are then stored in a list called gmap\_ids\_review\_fields. This filtered list is compared against the total list of gmap\_ids in the original dataset, to understand how much many businesses have more than 70 reviews.

```
[ ] # Filter the dataframe to include only rows where the number of text reviews ('review_text_count') is greater than 70
df_review_fields = df_csv[df_csv['review_text_count'] > 70]

# Extract the list of gmap_ids from the filtered dataframe
gmap_ids_review_fields = df_review_fields['gmap_id'].tolist()

# Extract the list of all gmap_ids from the original dataframe
gmap_ids_organial = df_csv['gmap_id'].tolist()

print(gmap_ids_review_fields[:5])
print()
print("Total Gmap_id's in original csv file:",len(gmap_ids_organial))
print("Total Gmap_id's with more than 70 text reviews:",len(gmap_ids_review_fields))
```

['0x8080354faaaaaab:0x12fa86bfac99deaf', '0x808158dba147f905:0x4bf9cfb6c3e27d6b', '0x808418adf88d73d7:0x9f193f3ea32a50', '0x8084388e140e3dbd:0x45b28a2f335dedb9',

Total Gmap\_id's in original csv file: 176  
Total Gmap\_id's with more than 70 text reviews: 82

Two dictionaries, extracted\_reviews and cleaned\_reviews, are initialized to store the review texts for each business (gmap\_id) that has more than 70 text reviews. The code iterates through the list of filtered gmap\_ids (gmap\_ids\_review\_fields) and, for each gmap\_id, retrieves the associated reviews from the json\_data. The review texts are extracted, excluding those labeled as 'None'. These extracted reviews are stored in the extracted\_reviews dictionary. To ensure the data is clean and usable for further analysis, any empty strings or None values are removed from the reviews before storing them in the cleaned\_reviews dictionary.

```
[ ] # Initialize dictionaries to store extracted and cleaned reviews
extracted_reviews = {}
cleaned_reviews = {}

# Iterate over the list of gmap_ids that have more than 70 text reviews
for gmap_id in gmap_ids_review_fields:
    if gmap_id in json_data:
        # Retrieve all reviews for the current gmap_id, defaulting to an empty list if none are found
        all_reviews = json_data[gmap_id].get('reviews', [])

        # Extract the review text for each review ignoring 'None'
        texts = [i.get('review_text') for i in all_reviews if i.get('review_text') != 'None']

        # Store the extracted reviews in the 'extracted_reviews' dictionary
        extracted_reviews[gmap_id] = texts

        # Further clean the reviews by removing any empty strings or None values, and store them in 'cleaned_reviews'
        cleaned_reviews[gmap_id] = [review for review in texts if review]
```

## Task 4: Data Preprocessing

Date: 25/08/2024 (Sun)

### Subtask 1: Tokenization

- **Description:**  
Tokenization was performed using a regular expression tokenizer to extract alphabetic tokens. Tokens with a length of less than 3 characters were removed to filter out less meaningful words.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Implemented the tokenizer to extract tokens and remove short tokens.

## **Subtask 2: Context-Independent Stopwords Removal**

- **Description:**  
Context-independent stopwords were removed from the tokenized data using a predefined list of common English stopwords.
- **Contributions:**
  - **Chaitanya Tambolkar:** Loaded the stopwords and removed them from the tokenized data.

## **Subtask 3: Context-Dependent Stopwords Removal**

- **Description:**  
Context-dependent stopwords (appearing in more than 95% of businesses) were identified and removed.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Identified high-frequency tokens and removed them from the tokenized data.

## **Subtask 4: Rare Tokens Removal**

- **Description:**  
Rare tokens (appearing in less than 5% of businesses) were identified and removed to ensure that the vocabulary was meaningful.
- **Contributions:**
  - **Chaitanya Tambolkar:** Filtered out rare tokens and updated the token list.

## **Subtask 5: Porter Stemming**

- **Description:**  
Porter Stemming was applied to reduce tokens to their root forms.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Applied the stemming algorithm to the cleaned tokens.

## **Subtask 6: Bigram Extraction**

- **Description:**  
Extracted the first 200 meaningful bigrams using the PMI measure and included them in the vocabulary.
- **Contributions:**



- **Chaitanya Tambolkar:** Performed bigram extraction and updated the vocabulary lists.

## Proof:

### ✓ 4.1. Tokenization

Tokenization is a principal step in text processing and producing unigrams. In this section, we applied a regular expression tokenizer to extract alphabetic tokens. Tokens with a length of less than 3 characters were removed to filter out less meaningful words.

```
[ ] # Tokenize and remove tokens less than length 3
tokenizer = RegexpTokenizer(r'[a-zA-Z]{3,}')

def tokenize_text(text):
    return tokenizer.tokenize(text)

tokenized_reviews = {}

for gmap_id, reviews in cleaned_reviews.items():
    tokenized_reviews[gmap_id] = [[token for token in tokenize_text(review)
                                   if len(token) >= 3] for review in reviews]
```

### ✓ 4.2. Context-Independent Stopwords Removal

In this section, we addressed context independent stopwords by utilizing a predefined list of common English stopwords. These words were loaded from an external file.

```
[ ] # Remove context independent Stop Words
import nltk

# Path to the stopwords file
stopwords_file_path = '/Users/fahmidtaawsifkhanchowdhury/Documents/Monash/S3/FIT5196/Assignment 1/stopwords_en.txt'

with open(stopwords_file_path, 'r', encoding='utf-8') as file:
    stopwords_set = set(file.read().strip().split('\n'))

# Display the number of stopwords loaded (optional)
print(f"Number of stopwords loaded: {len(stopwords_set)}")
```

➡ Number of stopwords loaded: 570

### ✓ 4.3. Context-Dependent Stopwords and Rare Tokens Removal

In this section, we addressed context dependent stopwords as well as rare tokens.

- Context dependent stopwords are the words that appear in more than 95% of the businesses that have at least 70 text reviews.
- Rare tokens are words that appear in less than 5% of the businesses that have at least 70 text reviews.

We define thresholds to determine which words are considered context-dependent stopwords and rare tokens. The max\_threshold is set to 95% of the businesses (with at least 70 text reviews) to identify words that appear too frequently across businesses, making them less informative. The min\_threshold is set to 5% to filter out words that appear too infrequently.

```
[ ] # Define the threshold levels for Context dependent stopwords and Rare tokens
max_threshold = 0.95 * int(len(gmap_ids_review_fields))
min_threshold = 0.05 * int(len(gmap_ids_review_fields))
```

For each business, the unique words across all its reviews are identified and stored as sets. These sets are then combined into a single list of words representing all businesses.

A frequency distribution (FreqDist) is created for the words, reflecting how many businesses each word appears in. This distribution helps in identifying words that are either too common (context-dependent stopwords) or too rare.

Words that appear in more than 95% of the businesses are identified as high-frequency tokens, which are context-dependent stopwords. Words appearing in fewer than 5% of businesses are considered rare tokens. Both sets of tokens are combined into a single set of tokens\_to\_remove.

```
[ ] # Identify high-frequency tokens (context-dependent stopwords)
    high_freq_tokens = {token for token, freq in frequency_words.items() if freq >= max_treshold}

    # Identify low-frequency tokens (rare tokens)
    low_freq_tokens = {token for token, freq in frequency_words.items() if freq <= min_treshold}

    # Combine the high-frequency and low-frequency tokens into a single set for removal
    tokens_to_remove = high_freq_tokens.union(low_freq_tokens)

    # Display the set
    # print("Context Dependent Stopwords:\n", tokens_to_remove)
```

The tokens are cleaned by iterating over each review and removing any tokens identified as either context-dependent stopwords or rare tokens. This results in a refined token list that excludes both extremely common and rare words.

```
[ ] cleaned_tokens = {}

    for gmap_id, reviews in filtered_tokens.items():
        cleaned_tokens[gmap_id] = [[token for token in review if token.lower() not in tokens_to_remove]
                                   for review in reviews]

    # cleaned_tokens
```

#### 4.4. Porter Stemming

The aim of this section is to apply the Porter Stemming algorithm to the tokenized review data and prepare a comprehensive list of all stemmed tokens.

```
[ ] # create an instance of the PorterStemmer
    stemmer = PorterStemmer()
```

The code iterates over the cleaned\_tokens dictionary. For each gmap\_id, the algorithm applies the Porter Stemmer to every token in the reviews, converting the tokens to their root forms. The stemmed tokens are then stored in a new dictionary called stemmed\_tokens, where each gmap\_id is associated with its corresponding list of stemmed reviews.

```
[ ] # Initialize an empty dictionary to store the stemmed tokens for each gmap_id
    stemmed_tokens = {}

    # Iterate over the tokenized reviews (with no stopwords) for each gmap_id
    for gmap_id, reviews in cleaned_tokens.items():
        # For each review, apply the Porter stemmer to each token in the review
        stemmed_tokens[gmap_id] = [
            [stemmer.stem(token) for token in review]
            for review in reviews]
```

#### 4.5. Bigrams

In this section, we extracted the first 200 meaningful bigrams that are included in the vocabulary using the PMI (Pointwise Mutual Information) measure.

```
[ ] # Initialize the Bigram Association Measures
    bigram_measures = nltk.collocations.BigramAssocMeasures()

    # Find bigrams in the list of all tokens using the BigramCollocationFinder
    finder = nltk.collocations.BigramCollocationFinder.from_words(all_tokens)

    # Select the top 200 bigrams based on PMI (Pointwise Mutual Information)
    top_200_bigrams = finder.nbest(bigram_measures.pmi, 200)

    # Join the bigram pairs with an underscore to create a single token
    bigrams = ['_'.join(i) for i in top_200_bigrams]
```

The vocab list is created by combining all unique unigrams (all\_tokens) and the top 200 bigrams.

```
[ ] vocab = sorted(set(all_tokens + bigrams))
```

The vocabulary is sorted and converted into a dictionary (vocab\_dict) where each vocabulary item is assigned a unique index.

```
[ ] vocab_dict = {item: index for index, item in enumerate(vocab)}
```

---

## Task 5: Writing Output Files

**Date:** 26/08/2024 (Monday)

### Description:

The final step involved exporting the cleaned and processed data. We generated the vocabulary list and the sparse matrix, which was then saved to text files.

### Contributions:

- **Team member 1: Fahmid Tawsif Khan Chowdhury**
  - Generated the vocabulary list and saved it as a text file.
- **Team member 2: Chaitanya Tambolkar**
  - Created the sparse matrix using CountVectorizer and exported it to a text file.

### Proof:

#### ✓ 5.1. Vocabulary List

This step involves saving the generated vocabulary dictionary to a specified file path for future use.

```
[ ] # Destination file path
    destination_path = '/Users/fahmidtawsifkhanchowdhury/Documents/Monash/S3/FIT5196/Assignment 1/Submissions'

    # Open the file in write mode and write the vocabulary dictionary
    with open(f'{destination_path}/130_vocab.txt', 'w') as f:
        for key, value in vocab_dict.items():
            f.write(f"{key}:{value}\n")
```

#### ✓ 5.2. Sparse Matrix

In this step, the tokenized and stemmed reviews are first flattened into a single list of tokens for each business (gmap\_id).

```
[ ] # Initialize an empty dictionary
    flattened_tokens = {}

    # Iterate through the stemmed tokens for each gmap_id
    for ids, reviews in stemmed_tokens.items():
        # Flatten the list of lists (reviews) into a single list of tokens for this gmap_id
        flattened_list = [token for review in reviews for token in review]
        # Store the flattened list in the dictionary
        flattened_tokens[ids] = flattened_list
```

A CountVectorizer is then initialized with the predefined vocabulary dictionary to map each token to a unique index. For each gmap\_id, the flattened list of tokens is transformed into a count vector, which records the frequency of each token. These count vectors are stored in a list.

```
[ ] # Initialize a CountVectorizer with the predefined vocabulary
    vectorizer = CountVectorizer(vocabulary=vocab_dict)

    # Initialize an empty list
    countvector = []

    # Iterate through the flattened tokens for each gmap_id
    for ids, review in flattened_tokens.items():
        # Join the tokens into a single string of text for each gmap_id
        total_text = ' '.join(review)
        # Transform the text into a count vector using the vectorizer
        total_vectors = vectorizer.transform([total_text])
        indices = total_vectors.indices # Get the indices of the tokens in the vocabulary
        counts = total_vectors.data # Get the counts of the tokens

        # Create a list of token indices and their counts in the format "index:count"
        token_indices = [
            f"{index}:{counts[i]}"
            for i, index in enumerate(indices) if counts[i] > 0
        ]
        # If there are any token counts, append them to the countvector list
        if token_indices:
            countvector.append(f"{ids}, " + ", ".join(token_indices))
```

The count vector list written to a text file in a specific format that associates each gmap\_id with its corresponding token counts. This output serves as a sparse representation of the reviews.

---

## Task 6: Summary

**Date: 26/08/2024 (Monday)**

### Description:

We documented the summary of the task, outlining the steps taken from data loading, preprocessing, to the generation of the final output files. This summary provides an overview of the processes and their significance.

### Contributions:

- **Team member 1: Fahmid Tawsif Khan Chowdhury**
  - Drafted the summary section, explaining each preprocessing step in detail.
- **Team member 2: Chaitanya Tambolkar**
  - Reviewed and provided feedback on the summary to ensure clarity and completeness.

## Proof:

### ✓ 6. Summary

- In this task, we successfully extracted and processed textual data from the output files of task 1.
- The process began with loading the data, followed by tokenization and the removal of irrelevant tokens based on - length and stopword criteria.
- Further refinement involved the removal of context-dependent stopwords and rare tokens, ensuring that the vocabulary was both meaningful and representative of the data.
- The data was then stemmed using the Porter Stemmer.
- Significant bigrams were identified and included in the final vocabulary.
- The clean and structured data was exported into a `.txt` format, and count vectors were generated and exported in `.txt` format as well.

## TASK 3 - Development History

### Student Names:

- Fahmid Tawsif Khan Chowdhury (Student ID: 34121315)
- Chaitanya Tambolkar (Student ID: 34093117)

**Date:** 27/08/2024

**Environment:** Python 3.11.5

### Libraries Used:

- pandas (for data manipulation)
- json (for reading and manipulating JSON files)
- matplotlib.pyplot (for data visualization)
- seaborn (for statistical data visualization)
- collections.Counter (for counting hashable objects)
- re (for regular expression operations)
- nltk (for natural language processing)
- nltk.sentiment.vader.SentimentIntensityAnalyzer (for sentiment analysis)

---

## Task 1: Introduction

**Date:** 27/08/2024

### Description:

The initial task involved understanding the objectives of Task 3 in Assessment 1 for FIT5196. The task required examining raw data, metadata, and conducting data analysis to answer specific business-related questions. The focus was on processing and analyzing the data to gain insights into various factors that impact business ratings.

### Proof:

- We closely followed Week 5 applied as well as (pdf related to EDA) uploaded on Moodle under Week 5 own time for exploring insights related to EDA for analysing the steps required for Task 3. The notes we found for those steps are as follows:

**Purpose of EDA:** Exploratory Data Analysis (EDA) is a critical first step in analysing data, primarily used for detecting mistakes, checking assumptions, selecting appropriate models, and determining relationships among variables.

- **Types of EDA:** EDA is generally categorized into non-graphical and graphical methods. Non-graphical methods involve summary statistics, while graphical methods summarize data visually. Additionally, EDA can be univariate (focusing on a single variable) or multivariate (exploring relationships between multiple variables).
- **Univariate Analysis:** This involves examining the distribution of a single variable, including its central tendency (mean, median), spread (variance, standard deviation), and potential outliers. Univariate graphical methods, like histograms and boxplots, are often used to visualize these characteristics.
- **Multivariate Analysis:** This explores relationships between two or more variables. Common graphical methods include scatterplots and side-by-side boxplots, which can reveal correlations and trends between variables.
- **Key EDA Techniques:** The document emphasizes histograms for understanding the distribution of data, boxplots for visualizing central tendency and spread, and scatterplots for examining relationships between variables.

---

### Task 2: Importing Libraries

**Date:** 27/08/2024

#### Description:

The necessary libraries were imported to accomplish the data analysis tasks. These libraries included pandas for data manipulation, json for handling JSON files, matplotlib.pyplot and seaborn for data visualization, collections.Counter for counting hashable objects, re for regular expression operations, and nltk for natural language processing.

#### Contributions:

- **Fahmid Tawsif Khan Chowdhury:** Imported the necessary libraries and set up the environment.
- **Chaitanya Tambolkar:** Reviewed the code and ensured that all required libraries were correctly imported.

### Proof:

```
[ ] import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import re
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

---

### Task 3: Examining Raw Data

**Date:** 28/08/2024

#### Description:


The raw data from Task 1 was loaded, and its structure was examined to understand the data types, summary statistics, and distribution of key variables.

#### Contributions:

- **Fahmid Tawsif Khan Chowdhury:** Loaded and explored the raw data, focusing on understanding its structure.
- **Chaitanya Tambolkar:** Conducted a thorough review of the summary statistics and distribution of key variables.

### Proof:

```
[ ] # Summary statistics for numerical columns
df_csv.describe()
```

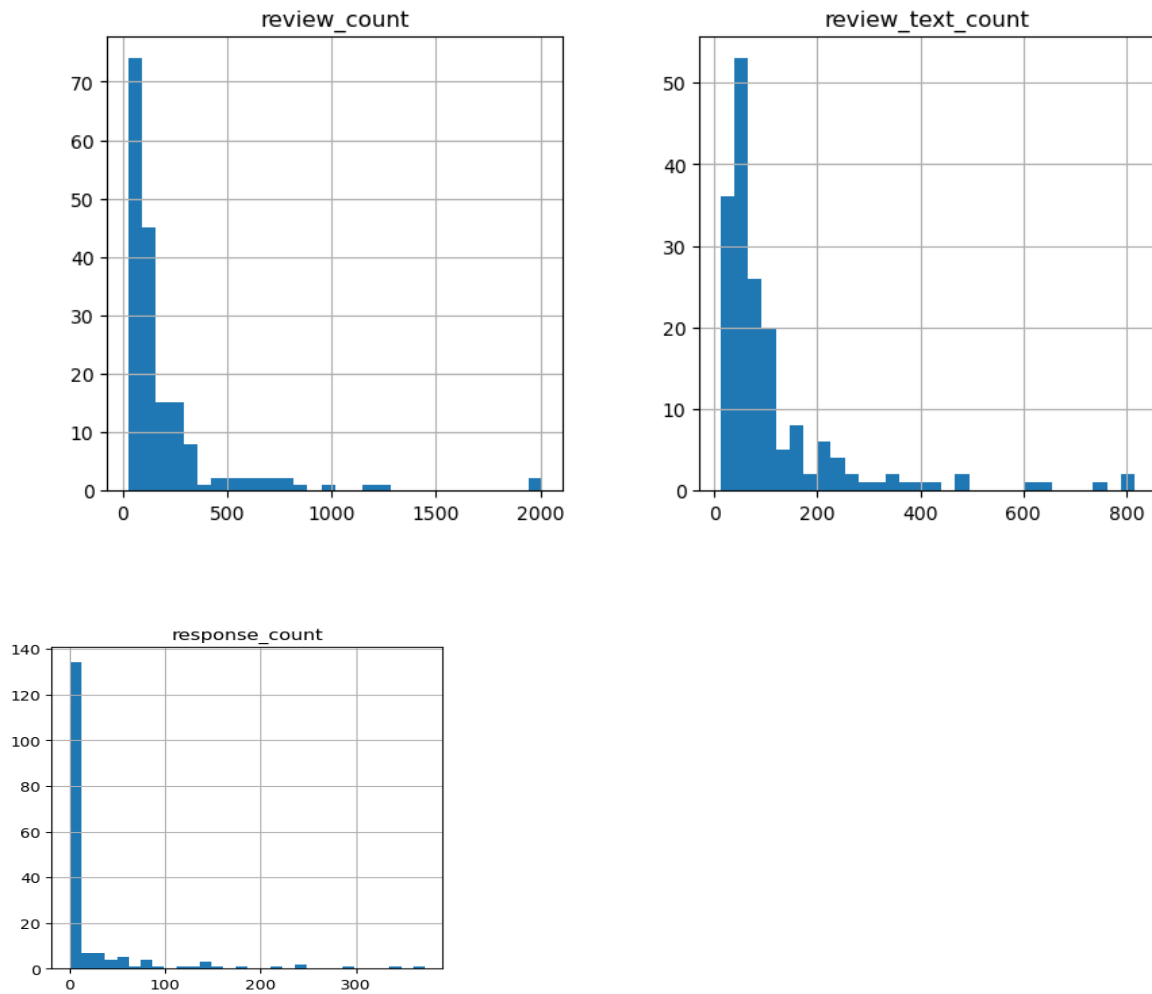


	review_count	review_text_count	response_count
count	176.000000	176.00000	176.000000
mean	201.744318	114.31250	23.403409
std	278.317593	137.09054	59.683947
min	26.000000	12.00000	0.000000
25%	68.000000	44.00000	0.000000
50%	108.000000	65.00000	0.000000
75%	202.000000	116.00000	10.000000
max	2009.000000	816.00000	372.000000

```
# Histograms
df_csv[['review_count', 'review_text_count', 'response_count']].hist(bins=30, figsize=(10, 10))
plt.suptitle("Histograms for Numerical Variables in df_csv")
plt.show()
```



Histograms for Numerical Variables in df\_csv



- The histogram for review\_count clearly shows a wide variation among businesses. While most businesses have fewer reviews, there are a few outliers with a significantly higher number of reviews. This is consistent with the high standard deviation of 278.32 compared to the mean of 201.74. The maximum number of reviews, reaching up to 2009, further highlights this disparity.
- The histogram for response\_count is heavily skewed to the right, with the majority of businesses having a response count of 0. The median value being 0 indicates that more than half of the businesses did not respond to any reviews. However, there are some businesses with very high engagement, with response counts reaching up to 372, indicating that a small subset of businesses is actively engaging with their customers.
- The histogram for review\_text\_count shows that while text reviews are common, their distribution is uneven. On average, about half of the reviews contain text, as indicated by the mean of 114.31 out of 201.74 average total reviews. The standard deviation of 137.09 and the wide range from 12 to 816 reviews indicate that while some businesses have most of their reviews containing text, others have very few.
- The distribution of review\_count is skewed, with a concentration of businesses having lower review counts. However, there is a significant number of businesses with much higher counts, as seen by the mean being closer to the 75th percentile than the median (201.74 mean vs. 108 median). This suggests a skew towards businesses with higher engagement.



```
# Correlation Analysis
correlation_matrix = df_csv[['review_count', 'review_text_count', 'response_count']].corr()
correlation_matrix
```

```

      review_count  review_text_count  response_count
review_count      1.000000      0.947333      0.128925
review_text_count  0.947333      1.000000      0.205173
response_count     0.128925      0.205173      1.000000

```

- The correlation between review\_count and review\_text\_count is 0.947, indicating a very strong positive relationship. This suggests that businesses with a higher total number of reviews also tend to have more text reviews. In other words, as the total number of reviews increases, the number of text reviews tends to increase proportionally.
- The correlation between review\_count and response\_count is 0.129, which indicates a weak positive relationship. This means that there is a slight tendency for businesses with more reviews to have more responses.
- Similarly, the correlation between review\_text\_count and response\_count is 0.205. This is also a weak positive correlation, suggesting that businesses with more text reviews are somewhat more likely to have more responses.

```
[ ] df_json.isnull().sum()
```

```

gmap_id      0
user_id      0
time         0
review_rating 0
review_text  0
if_pic       0
if_response  0
dtype: int64

```

```
[ ] df_json.describe()
```

```

      review_rating
count  35507.000000
mean    4.315572
std     1.169384
min     1.000000
25%     4.000000
50%     5.000000
75%     5.000000
max     5.000000

```

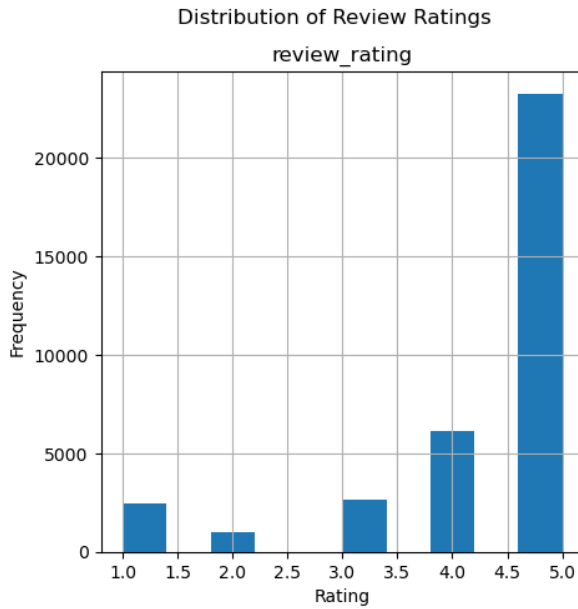
```
[ ] df_json.describe(include = 'O')
```

```

      gmap_id      user_id      time  review_text  if_pic  if_response
count      35507      35507      35507      35507      35507      35507
unique        176      35076      35500      19344         2         2
top  0x80c2d5884a05df65:0xd701a2f278b43cb0  100753780228758754876  2020-07-18 22:31:52      None         N         N
freq         2009         4         2      15388      34433      31388

```

17



- The distribution of review ratings is heavily skewed towards the higher end, with the majority of ratings being 4.0 and above.
- The mean rating is 4.32, indicating that the reviews are quite favorable. This suggests that most customers are satisfied with the service/product they are reviewing.
- There are relatively few reviews with ratings of 1.0 or 2.0, as shown in the histogram. This could indicate that either negative experiences are rare or it might also mean that people are less likely to leave a review if dissatisfied.

```
[ ] df_json.describe(include=['O'])
```

	gmap_id	user_id	time	review_text	if_pic	if_response
count	35507	35507	35507	35507	35507	35507
unique	176	35076	35500	19344	2	2
top	0x80c2d5884a05df65:0xd701a2f278b43cb0	100753780228758754876	2020-07-18 22:31:52	None	N	N
freq	2009	4	2	15388	34433	31388

- The most frequent gmap\_id appears 2,009 times, suggesting that a particular business is receiving a disproportionately high number of reviews. This could indicate a popular business.
- 35,076 out of 35,507 user\_ids were unique. This indicates that customers are not likely to review more than once.
- The most common entry for review\_text is "None", appearing 15,388 times. This indicates that a significant portion of the reviews (15,388 times out of 35,507) do not include any textual feedback.
- Only 34,433 entries indicate that no picture was included with the review, with "N" as the most frequent value. This shows that most users do not upload pictures alongside their reviews.
- "N" is the most common value for if\_response, occurring 31,388 times. This suggests that most reviews do not receive a direct response from the business. This is consistent with our previous findings.

## Task 4: Examining Metadata

Date: 28/08/2024

### Description:

The metadata associated with the businesses was examined to gather additional context for the analysis. This included understanding the format of fields such as category, hours, MISC, state, and relative\_results.

### Contributions:

- **Fahmid Tawsif Khan Chowdhury:** Processed and explored the metadata.
- **Chaitanya Tambolkar:** Validated the structure and consistency of the metadata with the raw data.

### Proof:

[ ] # Convert the list of metadata into a DataFrame df_meta = pd.DataFrame(meta_data)													
	name	address	gmap_id	description	latitude	longitude	category	avg_rating	num_of_reviews	price	hours	MISC	state
0	City Textile	City Textile, 3001 E Pico Blvd, Los Angeles, C...	0x80c2c98c0e3c16fd:0x29ec8a728764dfd9	None	34.018891	-118.215290	[Textile exporter]	4.5	6	None	None	None	Open now
1	San Soo Dang	San Soo Dang, 761 S Vermont Ave, Los Angeles, ...	0x80c2c778e3b73d33:0xbdc58662a4a97d49	None	34.058092	-118.292130	[Korean restaurant]	4.4	18	None	[[Thursday, 6:30AM–6PM], [Friday, 6:30AM–6PM],...	{'Service options': ['Takeout', 'Dine-in', 'De...	Open · Closes 6PM
2	Nova Fabrics	Nova Fabrics, 2200 E 11th St, Los Angeles, CA ...	0x80c2c89923b27a41:0x32041559418d447	None	34.023669	-118.232930	[Fabric store]	3.3	6	None	[[Thursday, 9AM–5PM], [Friday, 9AM–5PM], [Satu...	{'Service options': ['In-store shopping'], 'Pa...	Open · Closes 5PM
3	Nobel Textile Co	Nobel Textile Co, 719 E 9th St, Los Angeles, C...	0x80c2c632f933b073:0xc31785961fe826a6	None	34.036694	-118.249421	[Fabric store]	4.3	7	None	[[Thursday, 9AM–5PM], [Friday, 9AM–5PM], [Satu...	{'Service options': ['In-store pickup']}	Open · Closes 5PM
4	Matrix International Textiles	Matrix International Textiles, 1363 S Bonnie B...	0x80c2cf163db6bc89:0x219484e2edbcfa41	None	34.015505	-118.181839	[Fabric store]	3.5	6	None	[[Thursday, 8:30AM–5:30PM], [Friday, 8:30AM–5...	{'Accessibility': ['Wheelchair accessible entr...	Open · Closes 5:30PM

MISC	state	relative_results	url
None	Open now	[0x80c2c624136ea88b:0xb0315367ed448771, 0x80c2...	https://www.google.com/maps/place//data=!4m2!3...
{'Service options': ['Takeout', 'Dine-in', 'De...	Open · Closes 6PM	[0x80c2c78249aba68f:0x35bf16ce61be751d, 0x80c2...	https://www.google.com/maps/place//data=!4m2!3...
{'Service options': ['In-store shopping'], 'Pa...	Open · Closes 5PM	[0x80c2c8811477253f:0x23a8a492df1918f7, 0x80c2...	https://www.google.com/maps/place//data=!4m2!3...
{'Service options': ['In-store pickup']}	Open · Closes 5PM	[0x80c2c62c496083d1:0xdefa11317fe870a1, 0x80c2...	https://www.google.com/maps/place//data=!4m2!3...
{'Accessibility': ['Wheelchair accessible entr...	Open · Closes 5:30PM	[0x80c2cf042a5d9561:0xd0024ad6f81f1335, 0x80c2...	https://www.google.com/maps/place//data=!4m2!3...

```
[ ] # Check for missing values and data types
print(df_meta.isna().sum())
print(df_meta.dtypes)
```

```
name          10
address       7704
gmap_id        0
description    404040
latitude       0
longitude      0
category      2376
avg_rating     0
num_of_reviews 0
price         406160
hours         100979
MISC          82923
state         143713
relative_results 40464
url           0
dtype: int64
name          object
address       object
gmap_id       object
description    object
latitude      float64
longitude     float64
category      object
avg_rating    float64
num_of_reviews int64
price         object
hours         object
MISC          object
state         object
relative_results object
url           object
dtype: object
```

After analyzing the auxiliary metadata dataset, we found that it includes valuable information such as the business name, address, category, and other relevant details. These data points are necessary for a more comprehensive analysis. By joining this metadata with our task 1 datasets using the gmap\_id as the key, we can create a new enriched dataframe that will enhance the depth and quality of our subsequent analysis.

Upon further examining the meta data structure, we found that the structure of the data fields exhibits a variety of formats.

- The category column is presented as a list of strings.
- The hours column is structured as a nested list.
- The MISC column is a dictionary where the keys represent different aspects of the business, and the values are lists that provide specific details.
- The state column can vary between different formats such as 'Open now', 'Open · Closes 8PM', or 'permanently closed'.
- The relative\_results column is a list of strings.

## review rating

```
[ ] # Calculate the count of responses ('Y' and 'N') for each review rating
response_counts = df_json.groupby(['review_rating', 'if_response']).size().unstack().fillna(0)

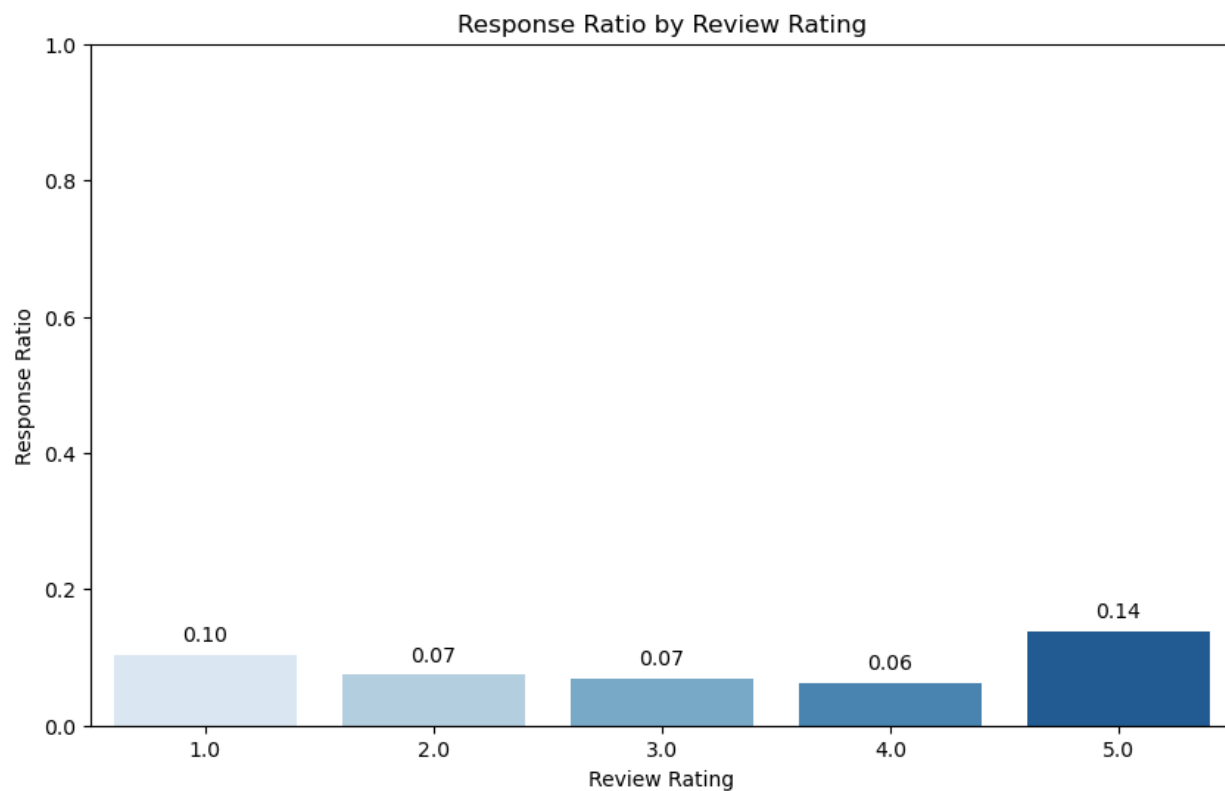
# Calculate the ratio of 'Y' responses to the total number of responses ('Y' + 'N') for each review rating
response_ratio = response_counts['Y'] / (response_counts['Y'] + response_counts['N'])

# Plot the ratio
plt.figure(figsize=(10, 6))
sns.barplot(x=response_ratio.index, y=response_ratio.values, palette='Blues')

# Add data labels
for index, value in enumerate(response_ratio.values):
    plt.text(x=index, y=value + 0.02, s=f"{value:.2f}", ha='center')

# Set plot title and labels
plt.title('Response Ratio by Review Rating')
plt.xlabel('Review Rating')
plt.ylabel('Response Ratio')
plt.ylim(0, 1)

# Show the plot
plt.show()
```



## Task 5: Data Processing and Analysis

**Date:** 29/08/2024 - 15/08/2024

### Description:

The data processing and analysis were divided into several subtasks to address the key questions and derive insights.

### Subtasks and Contributions:

#### 1. Subtask 1: Data Merging and Enrichment

- **Date:** 06/08/2024
- **Description:** Merged the JSON reviews with metadata and the CSV data to create a comprehensive dataset.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Led the data merging process.
  - **Chaitanya Tambolkar:** Reviewed and verified the merged data for accuracy.

### Proof:

```
[ ] # Remove duplicates from df_meta
df_meta_clean = df_meta.drop_duplicates(subset=['gmap_id'])
```

```
[ ] # Merge JSON reviews with metadata
merged_json_meta = pd.merge(df_json, df_meta_clean, on='gmap_id', how='left')

# Merge the above result with the CSV data
df_csv_json_meta = pd.merge(merged_json_meta, df_csv, on='gmap_id', how='left')
```

```
[ ] df_csv_json_meta.head(2)
```

	gmap_id	user_id	time	review_rating	review_text	if_pic	if_response	name	address	description	...	num_of_reviews	price	hours	MISC	state
0	0x54d29333b3ff25830xa4c448dcc281f718	106152623296557720501	2021-05-15 01:02:20	5.0	worked me in for a quick haircut when they wer...	N	N	Supercuts	Supercuts, 1150 Dana Dr Ste 2, Redding, CA 96003	None	...	87	None	[[Tuesday, 9AM-7PM], [Wednesday, 9AM-7PM], [Th...	{'Accessibility': '[Wheelchair accessible entr...	Closed - Opens 9AM
1	0x54d29333b3ff25830xa4c448dcc281f718	104286580367132707943	2021-05-08 02:34:12	5.0	had the best experience. great haircut with e...	N	N	Supercuts	Supercuts, 1150 Dana Dr Ste 2, Redding, CA 96003	None	...	87	None	[[Tuesday, 9AM-7PM], [Wednesday, 9AM-7PM], [Th...	{'Accessibility': '[Wheelchair accessible entr...	Closed - Opens 9AM

2 rows x 24 columns

```
[ ] # Convert the 'time' column to datetime
df_csv_json_meta['time'] = pd.to_datetime(df_csv_json_meta['time'])
```

```
[ ] # Extract time features for possible analysis
df_csv_json_meta['hour'] = df_csv_json_meta['time'].dt.hour
df_csv_json_meta['day_of_week'] = df_csv_json_meta['time'].dt.day_name()
df_csv_json_meta['month'] = df_csv_json_meta['time'].dt.month
df_csv_json_meta['year'] = df_csv_json_meta['time'].dt.year
```

```
[ ] df_csv_json_meta.head(2)
```

	gmap_id	user_id	time	review_rating	review_text	if_pic	if_response	name	address	description	...	state	relative_results
0	0x54d29333b3ff25830xa4c448dcc281f718	106152623296557720501	2021-05-15 01:02:20	5.0	worked me in for a quick haircut when they wer...	N	N	Supercuts	Supercuts, 1150 Dana Dr Ste 2, Redding, CA 96003	None	...	Closed - Opens 9AM	[0x54d293369b47ff63:0x960597624a95221a, 0x54d2...
1	0x54d29333b3ff25830xa4c448dcc281f718	104286580367132707943	2021-05-08 02:34:12	5.0	had the best experience. great haircut with e...	N	N	Supercuts	Supercuts, 1150 Dana Dr Ste 2, Redding, CA 96003	None	...	Closed - Opens 9AM	[0x54d293369b47ff63:0x960597624a95221a, 0x54d2...

2 rows x 28 columns

## 2. Subtask 2: Sentiment Analysis

- **Date:** 08/08/2024
- **Description:** Performed sentiment analysis on the review text to calculate sentiment scores.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Implemented the sentiment analysis using NLTK.
  - **Chaitanya Tambolkar:** Analyzed and interpreted the sentiment scores.

### Proof:

```
[ ] sid = SentimentIntensityAnalyzer()

# Function to calculate sentiment score
def get_sentiment_score(text):
    sentiment_dict = sid.polarity_scores(text)
    return sentiment_dict['compound'] # -1 (most negative) and 1 (most positive)

# Create a new column 'sentiment_score'
df_csv_json_meta['sentiment_score'] = df_csv_json_meta['review_text'].apply(get_sentiment_score)

# Display the first few rows
df_csv_json_meta[['review_text', 'sentiment_score']].head(2)
```

	review_text	sentiment_score
0	worked me in for a quick haircut when they wer...	0.8516
1	had the best experience. great haircut with e...	0.8715

```
[ ] # Create review length column
df_csv_json_meta['review_length'] = df_csv_json_meta['review_text'].apply(len)
```

```
[ ] # Create a df for analysis
df_selected = df_csv_json_meta[['gmap_id', 'sentiment_score', 'review_length', 'review_rating', 'day_of_week', 'time_of_day', 'avg_rating', 'review_count', 'review_text_count', 'response_count', 'normalized_price']]
df_selected.head(5)
```

	gmap_id	sentiment_score	review_length	review_rating	day_of_week	time_of_day	avg_rating	review_count	review_text_count	response_count	normalized_price
0	0x54d2933b3ff2583:0xa4c448dcc281f718	0.8516	92	5.0	Saturday	Late Night	4.1	87	46	8	None
1	0x54d2933b3ff2583:0xa4c448dcc281f718	0.8715	91	5.0	Saturday	Late Night	4.1	87	46	8	None
2	0x54d2933b3ff2583:0xa4c448dcc281f718	-0.9200	596	1.0	Saturday	Evening	4.1	87	46	8	None
3	0x54d2933b3ff2583:0xa4c448dcc281f718	0.8591	48	5.0	Friday	Evening	4.1	87	46	8	None
4	0x54d2933b3ff2583:0xa4c448dcc281f718	-0.1531	136	1.0	Friday	Late Night	4.1	87	46	8	None

```
[ ] Start reading on pandas with dt
```

```
[ ] # Group by 'if_response' and calculate the average 'review_rating', 'sentiment_score', and 'review_length'
response_stats = df_csv_json_meta.groupby('if_response').agg({'review_rating': 'mean', 'sentiment_score': 'mean', 'review_length': 'mean'})

# Display the calculated averages
print(response_stats)
```

	if_response	review_rating	sentiment_score	review_length
0	N	4.289537	0.277617	86.137250
1	Y	4.513960	0.438215	161.488225

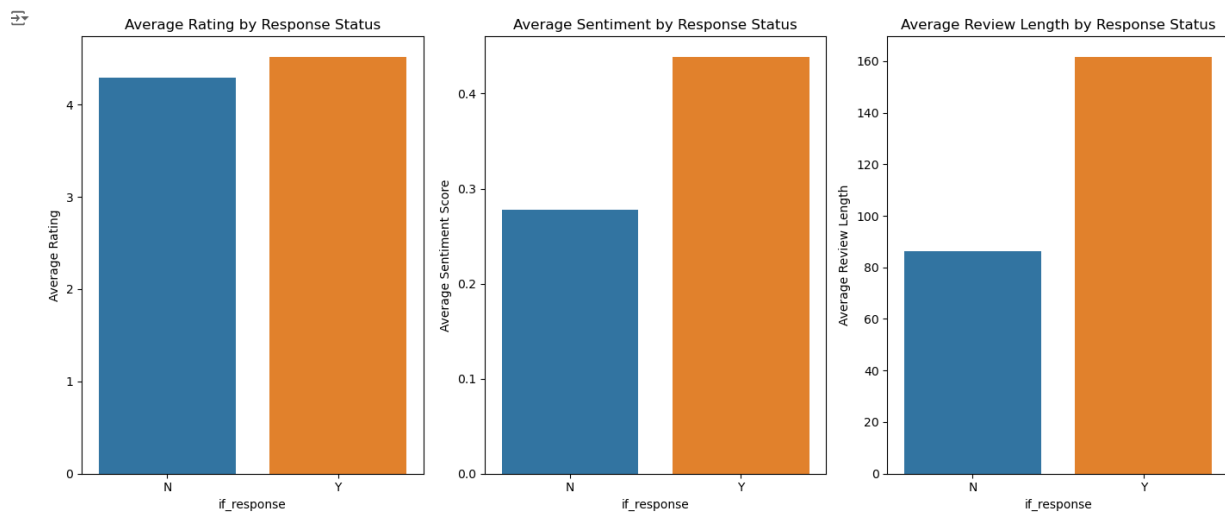
```
[ ] # Create bar charts for comparison
plt.figure(figsize=(14, 6))

# Bar chart for average review rating
plt.subplot(1, 3, 1)
sns.barplot(x='if_response', y='review_rating', data=response_stats)
plt.title('Average Rating by Response Status')
plt.ylabel('Average Rating')

# Bar chart for average sentiment score
plt.subplot(1, 3, 2)
sns.barplot(x='if_response', y='sentiment_score', data=response_stats)
plt.title('Average Sentiment by Response Status')
plt.ylabel('Average Sentiment Score')

# Bar chart for average review length
plt.subplot(1, 3, 3)
sns.barplot(x='if_response', y='review_length', data=response_stats)
plt.title('Average Review Length by Response Status')
plt.ylabel('Average Review Length')

plt.tight_layout()
plt.show()
```



- Reviews that received a response from the business tend to have higher average ratings. This suggests that engaging with customers could positively influence their satisfaction or at least correlate with higher satisfaction levels. It could also indicate that businesses are more likely to respond to higher-rated reviews, although this would require further investigation to confirm.
- There is a notable increase in the sentiment score when a review receives a response. Higher sentiment scores indicate more positive language in the review text, suggesting that responses may encourage or correlate with more positive sentiment. This might reflect that customers feel more valued when businesses engage with them, leading to more positive feedback.
- Reviews that received a response are almost twice as long as those that did not. This can suggest two things:
  - Businesses are more likely to respond to more detailed reviews.
  - The act of responding might encourage customers to leave more detailed feedback in the first place.



### Subtask 3: Price Normalization and Exploratory Data Analysis (EDA)

- Conducted EDA to explore the relationships between various factors, such as review count, response count, and sentiment score and normalized the price data to standardize currency symbols across the dataset.
- **Contributions:**
  - **Fahmid Tawsif Khan Chowdhury:** Performed the EDA and visualized the results and handled the normalization of price data.
  - **Chaitanya Tambolkar:** Provided insights and interpretations of the EDA findings and ensured the consistency of price data post-normalization.

#### Proof:

```
[ ] df_csv_json_meta['price'].unique()
```

```
⇒ array([None, '$$', '$', '₩', '₩₩', '$$$'], dtype=object)
```

```
[ ] # Function to normalise price
def normalize_price(price):
    if price == '₩':
        return '$'
    elif price == '₩₩':
        return '$$'
    else:
        return price

# Apply the function to create a new column 'normalized_price'
df_csv_json_meta['normalized_price'] = df_csv_json_meta['price'].apply(normalize_price)

# Check results
df_csv_json_meta[df_csv_json_meta['price'].isin(['₩', '₩₩'])][['price', 'normalized_price']].head(2)
```

```
⇒
```

	price	normalized_price
6221	₩	\$
6222	₩	\$

```
[ ] df_price = df_csv_json_meta[['normalized_price', 'review_count', 'review_text_count', 'response_count', 'avg_rating']]

df_price = df_price.drop_duplicates()

# Display the first few rows of the new DataFrame to verify
df_price.head(2)
```

```
⇒
```

	normalized_price	review_count	review_text_count	response_count	avg_rating
0	None	87	46	8	4.1
87	\$\$	358	223	0	4.4

```
[ ] df_price = df_csv_json_meta[['normalized_price', 'review_count', 'review_text_count', 'response_count', 'avg_rating']]

df_price = df_price.drop_duplicates()

# Display the first few rows of the new DataFrame to verify
df_price.head(2)
```

```

normalized_price  review_count  review_text_count  response_count  avg_rating
0              None           87                46                8          4.1
87             $$           358                223                0          4.4
```

```
[ ] # Convert 'price' to a categorical variable explicitly as strings
df_price['normalized_price'] = df_price['normalized_price'].astype(str)

# Treat dollar sign as string and not special character
df_price['normalized_price'] = df_price['normalized_price'].apply(lambda x: x.replace('$', r'\$') if isinstance(x, str) else x)
```

```
[ ] # Average review_count by price category
plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
sns.barplot(x='normalized_price', y='review_count', data=df_price, ci=None, palette="Blues_d")
plt.title('Average Review Count by Price')
plt.ylabel('Average Review Count')
plt.xlabel('Price Category')
```

`/var/folders/zq/j34847wd1dd19h01d3tqzb_40000gn/T/ipykernel_18892/683431883.py:5: FutureWarning:`

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

```
sns.barplot(x='normalized_price', y='review_count', data=df_price, ci=None, palette="Blues_d")
Text(0.5, 0, 'Price Category')
```

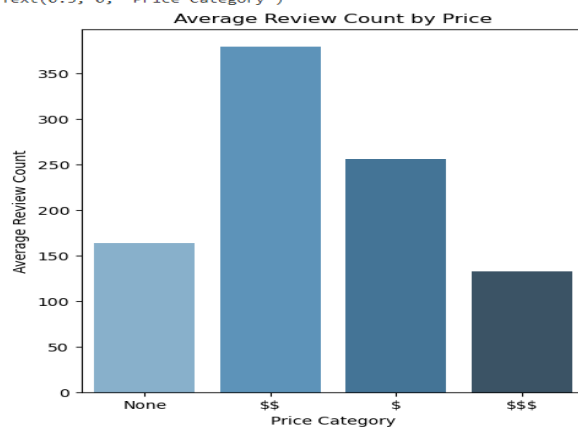
```
[ ] # Average review_count by price category
plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
sns.barplot(x='normalized_price', y='review_count', data=df_price, ci=None, palette="Blues_d")
plt.title('Average Review Count by Price')
plt.ylabel('Average Review Count')
plt.xlabel('Price Category')
```

`/var/folders/zq/j34847wd1dd19h01d3tqzb_40000gn/T/ipykernel_18892/683431883.py:5: FutureWarning:`

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

```
sns.barplot(x='normalized_price', y='review_count', data=df_price, ci=None, palette="Blues_d")
Text(0.5, 0, 'Price Category')
```



- The '\$\$' price category has the highest average review count compared to other price categories. This suggests that mid-range priced businesses attract the most customer engagement in terms of reviews. Consumers may feel that mid-range options offer the best balance of value and quality, prompting them to share their experiences more frequently. This could also indicate that mid-range businesses are more accessible to a broader audience, leading to higher customer interactions and as a result, more reviews.
- The '\$\$\$' price category, representing higher-priced businesses, has the lowest average review count. This suggests that high-end businesses might receive less customer feedback, possibly due to a smaller, more exclusive customer base. Customers at higher price points might have different expectations or be less inclined to leave reviews, or these businesses might serve fewer customers overall, leading to fewer opportunities for reviews. This trend highlights the possibility that as the price increases, customer engagement in terms of reviews tends to decrease.

```
[ ] # Filter the data for 2021
df_2021 = df_csv_json_meta[df_csv_json_meta['year'] == 2021]

# Calculate the number of reviews per month in 2021
reviews_per_month_2021 = df_2021['time'].dt.month.value_counts().sort_index()

# Calculate the average number of reviews per month in 2021
avg_reviews_per_month = reviews_per_month_2021.mean()

# Define the number of missing months for which data is not available
missing_months = 4

# Estimate the number of reviews for the missing months based on the average reviews per month
estimated_reviews_missing_months = avg_reviews_per_month * missing_months

# Calculate the actual total number of reviews in 2021
actual_reviews_2021 = reviews_per_month_2021.sum()

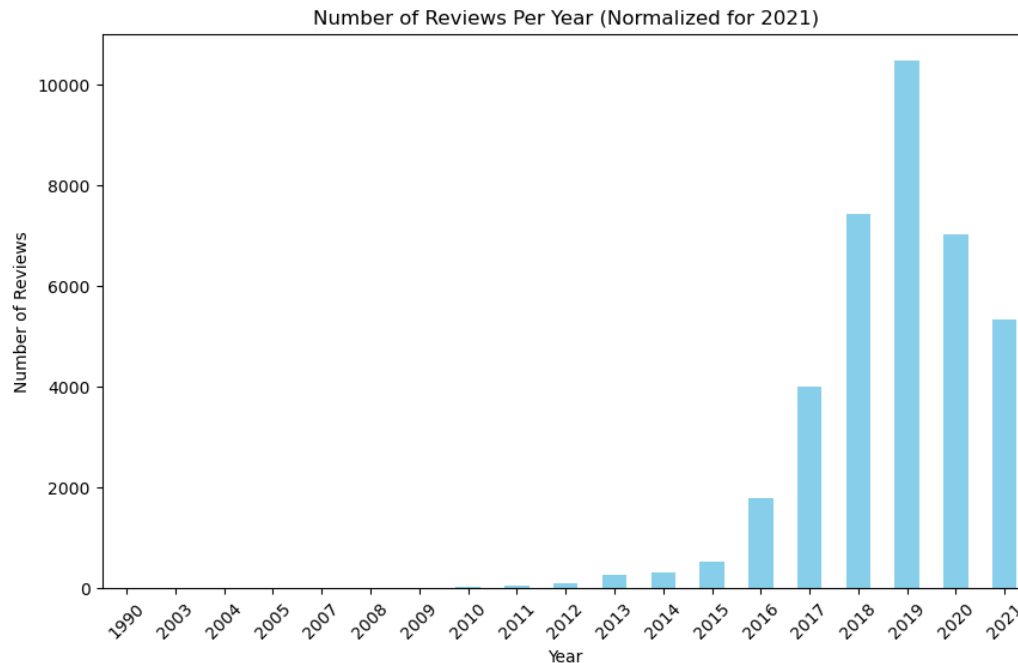
# Normalize the total number of reviews for 2021 by adding the estimated reviews for the missing months
normalized_reviews_2021 = actual_reviews_2021 + estimated_reviews_missing_months

# Make a copy of the reviews_per_year data to avoid modifying the original data
reviews_per_year_copy = reviews_per_year.copy()

# Update the 2021 data in the copied DataFrame with the normalized number of reviews
reviews_per_year_copy.loc[2021] = normalized_reviews_2021

# Sort the data by year to ensure chronological order in the plot
reviews_per_year_copy = reviews_per_year_copy.sort_index()

# Plot the updated bar chart using the copied and updated data
plt.figure(figsize=(10, 6))
reviews_per_year_copy.plot(kind='bar', color='skyblue')
plt.title('Number of Reviews Per Year (Normalized for 2021)')
plt.xlabel('Year')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=45)
plt.show()
```



- The number of reviews increased sharply from 2016 to 2019, reflecting a period of substantial growth in consumer engagement with businesses in California. This surge in reviews likely correlates with an increase in business interactions and visits during this period, such as more people were dining out, visiting stores, and participating in various other activities. The steady rise in reviews suggests a flourishing business environment where consumer activity were high.
- The steep drop in the number of reviews in 2020 is a clear indicator of the impact that the COVID-19 pandemic had on business interactions. The mandatory statewide stay-at-home order, along with the closure or suspension of non-essential businesses such as restaurants, movie theaters, and retail stores, led to a significant reduction in customer visits. This directly resulted in fewer opportunities for consumers to leave reviews which is an indication of how the pandemic severely disrupted normal business operations and consumer behavior.

---

## Task 6: Summary and Reporting

**Date:** 29/08/2024

### Description:

The findings from the data analysis were summarized, and a detailed report was prepared. The summary highlighted key insights, such as the correlation between review count and response count, the impact of sentiment scores, and the effect of price normalization on the analysis.

### Contributions:

- **Fahmid Tawsif Khan Chowdhury:** Drafted the summary and key findings.
- **Chaitanya Tambolkar:** Reviewed the summary and contributed additional insights.

## Proof:

### 6. Summary

Based on the analysis above, we investigated the following questions:

#### How does the review rating affect the likelihood of a business response?

- Our findings: Higher-rated reviews (5 stars) have a slightly higher response rate, though overall response rates are low across all rating categories. Businesses tend to engage more with extremely positive or extremely negative reviews.
- These findings tell us: Businesses may prioritize their engagement strategies based on the extremity of feedback, leaving room for significant improvement.

#### What is the relationship between the sentiment of review text and business responses?

- Our findings are: Reviews with higher sentiment scores, indicating more positive language, are more likely to receive a response from the business. Additionally, reviews that receive a response tend to be longer and more detailed.
- These findings tell us: Positive customer experiences are more likely to prompt business engagement and businesses may be encouraging more detailed feedback through their responses.

#### What was the impact of COVID-19 on the volume of reviews and business interactions?

- Our findings are: There was a significant decrease in the number of reviews in 2020 due to the COVID-19 pandemic. This reflects the direct impact of the pandemic on consumer behavior and business operations.
- These findings tell us: The pandemic severely disrupted normal business operations and consumer behavior, leading to reduced opportunities for customer reviews and engagement during lockdown periods.

#### How do the highest-rated businesses compare to the lowest-rated ones in terms of customer satisfaction?

- Our findings are: The top 10 businesses by average rating tend to be in sectors that offer personalized services, such as healthcare and repair services, while the lowest-rated businesses are often in high-volume, low-touch sectors like postal services and fast food.
- These findings tell us: Personalized, high-quality customer service is crucial in achieving high customer satisfaction, whereas businesses with less focus on individual customer experiences may struggle with lower ratings.

#### What relationship exists between price categories and customer review activity?

- Our findings are: Mid-range (\$\$) priced businesses receive the highest number of reviews, while higher-priced businesses (\$\$\$) tend to have fewer reviews. This suggests that mid-range businesses are more accessible and engage a broader customer base.
- 

#### What relationship exists between price categories and customer review activity?

- Our findings are: Mid-range (\$\$) priced businesses receive the highest number of reviews, while higher-priced businesses (\$\$\$) tend to have fewer reviews. This suggests that mid-range businesses are more accessible and engage a broader customer base.
- These findings tell us: Price sensitivity plays a role in customer engagement, with mid-range businesses being more likely to receive customer feedback, possibly due to their balance of value and quality. Higher-end businesses may cater to a smaller, more exclusive customer base, resulting in fewer reviews.