SIT742_Modern_Data_Science_

September 17, 2025

1 SIT742: Modern Data Science

Trimester 2, 2025

Assignment 2

Students:

- Jay Rahul Pawar (ID s224970276)
- Chudi Onwuakaike (ID)
- Julio Leon (ID 224504987)

2 *Part I*

2.1 Question 1.1

[1]:
```
# Setting up the collab for Spark -
!apt-get -y -qq install openjdk-11-jdk-headless
!pip -q install pyspark==3.5.1 findspark
```

[2]:
```
''' Findspark is the Python library which helps locate the Apache Spark when
running outside the Spark environment. Without findspark, Python might not
know where exactly Spark is installed.'''

# pyspark is the official Python API for Apache Spark

import os
import findspark, pyspark

findspark.init() # the init method sets up the environment
```

[3]:
```
from pyspark.sql import SparkSession
# SparkSession is the enrty point for programming with SparkSQL and DataFrames

spark = SparkSession.builder.appName("SIT742Assignment2").getOrCreate()
# The app name is given to monitor the application in Spark UI

spark
```

[3]:
```
<pyspark.sql.session.SparkSession at 0x7e5c92f3b3e0>
```

[4]:
```
!pip install -q wget
```

```
  Preparing metadata (setup.py) … done
  Building wheel for wget (setup.py) … done
```

[5]:
```
import zipfile, os, wget

zip_url = 'https://raw.githubusercontent.com/tulip-lab/sit742/refs/heads/
 ↪develop/Jupyter/data/business_review_submission.zip'
zip_file = wget.download(zip_url)

with zipfile.ZipFile(zip_file, 'r') as zf:
    zf.extractall('/content/data')  # choose folder

os.listdir('/content/data')  # check extracted files
```

[5]:
```
['review.csv', 'meta-review-business.csv']
```

[6]:
```
# Defining the path to the downloaded files from the github repo.
reviews_path = "data/review.csv"
meta_path = "data/meta-review-business.csv"

# Making two different Dataframes for the data files in the path.
# header -> true will infer that the first row is the header of the csv file.
# inferSchema -> true will tell Spark to guess the data types of the columns.
reviews_df = (
    spark.read
         .option("header", True)
         .option("inferSchema", True)
         .option("multiLine", True)                # reviews can span lines
         .option("quote", '"')                     # default, but explicit
         .option("escape", '"')                    # escape embedded quotes
         .option("ignoreLeadingWhiteSpace", True)
         .option("ignoreTrailingWhiteSpace", True)
         .option("mode", "PERMISSIVE")             # keep bad rows instead of␣
 ↪failing
         .option("columnNameOfCorruptRecord", "_corrupt_record")
         .csv(reviews_path)
)

meta_df = (
    spark.read
         .option("header", True)
         .option("inferSchema", True)
         .option("multiLine", True)
```

```
        .option("quote", '"')
        .option("escape", '"')
        .option("ignoreLeadingWhiteSpace", True)
        .option("ignoreTrailingWhiteSpace", True)
        .option("mode", "PERMISSIVE")
        .csv(meta_path)
)
```

[7]:
```
print("Reviews Schema: ")
reviews_df.printSchema()
reviews_df.show(5)
```

```
Reviews Schema:
root
 |-- user_id: double (nullable = true)
 |-- name: string (nullable = true)
 |-- time: long (nullable = true)
 |-- rating: integer (nullable = true)
 |-- text: string (nullable = true)
 |-- pics: string (nullable = true)
 |-- resp: string (nullable = true)
 |-- gmap_id: string (nullable = true)

+--------------------+----------------+-------------+------+--------------------
-+----+----+--------------------+
|             user_id|            name|         time|rating|
text|pics|resp|             gmap_id|
+--------------------+----------------+-------------+------+--------------------
-+----+----+--------------------+
|1.091298048426862E20|       Nicki Gore|1566331951619|     5|We always stay
he…|NULL|NULL|0x56b646ed2220b77…|
|1.132409264057589…|    Allen Ratliff|1504917982385|     5|Great campground
…|NULL|NULL|0x56b646ed2220b77…|
|1.130448378911412…|Jonathan Tringali|1474765901185|     4|We tent camped
he…|NULL|NULL|0x56b646ed2220b77…|
|1.103291551475920…|           S Blad|1472858535682|     4|This place is
jus…|NULL|NULL|0x56b646ed2220b77…|
| 1.08989634908602E20|   Daniel Formoso|1529649811341|     5|Probably the
nice…|NULL|NULL|0x56b646ed2220b77…|
+--------------------+----------------+-------------+------+--------------------
-+----+----+--------------------+
only showing top 5 rows
```

[8]:
```
print("Meta Schema: ")
meta_df.printSchema()
meta_df.show(5)
```

```
Meta Schema:
root
 |-- name: string (nullable = true)
 |-- address: string (nullable = true)
 |-- gmap_id: string (nullable = true)
 |-- description: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- category: string (nullable = true)
 |-- avg_rating: double (nullable = true)
 |-- num_of_reviews: integer (nullable = true)
 |-- price: string (nullable = true)
 |-- hours: string (nullable = true)
 |-- MISC: string (nullable = true)
 |-- state: string (nullable = true)
 |-- relative_results: string (nullable = true)
 |-- url: string (nullable = true)

+--------------------+--------------------+--------------------+-----------+----
--------------+-------------------+--------------------+----------+-------------
-+-----+--------------------+--------------------+--------------------+---------
-----------+--------------------+
|                name|             address|             gmap_id|description|
latitude|          longitude|
category|avg_rating|num_of_reviews|price|               hours|
MISC|               state|    relative_results|                 url|
+--------------------+--------------------+--------------------+-----------+----
--------------+-------------------+--------------------+----------+-------------
-+-----+--------------------+--------------------+--------------------+---------
-----------+--------------------+
|Bear Creek Cabins…|Bear Creek Cabins…|0x56b646ed2220b77…|       NULL|
61.1006437|-146.21455179999998|['RV park', 'Cabi…|       4.5|            18|
NULL|                NULL|                NULL|
NULL|['0x56b6445fd9f9e…|https://www.googl…|
|    Anchorage Market|Anchorage Market,…|0x56c8992b5dee722…|       NULL|
61.1414349|       -149.8684816| ["Farmers' market"]|       4.2|            18|
NULL|[['Thursday', 'Cl…|{'Service options…|Closed  Opens 10…|
NULL|https://www.googl…|
|     Happy Camper RV|Happy Camper RV, …|0x56c8e0455225be8…|
NULL|61.591855499999994|       -149.2906566|  ['RV repair shop']|       4.4|
28| NULL|                NULL|{'Accessibility':…|
NULL|['0x56c8e104d9929…|https://www.googl…|
|        Cajun Corner|Cajun Corner, 302…|0x56c8bdb5d91017c…|
NULL|61.219378299999995|       -149.8958522|['American restau…|       4.5|
24| NULL|[['Wednesday', '1…|{'Service options…|Closed  Opens 11…|
NULL|https://www.googl…|
|Alaska General Se…|Alaska General Se…|0x540c25195639567…|
NULL|55.336118799999994|       -131.6306694|['Seafood wholesa…|       4.7|
```

```
8| NULL|[['Wednesday', '7…|                NULL|  Open   Closes
11PM|['0x540c25a882a72…|https://www.googl…|
+--------------------+--------------------+--------------------+----------+----
-------------+------------------+--------------------+----------+-------------
-+-----+--------------------+--------------------+--------------------+---------
----------+--------------------+
only showing top 5 rows
```

2.1.1 Question 1.1.1

Cleaning none or null in text column, changing it to ''No review".

[9]:
```
from pyspark.sql.functions import col, sum

# Identifying nulls on each one of the columms reviews dataset
null_counts_reviews_df = reviews_df.select([
    sum(col(c).isNull().cast("int")).alias(c) for c in reviews_df.columns])

print("Number of Null valuess per column in  reviews Dataset \n")
null_counts_reviews_df.show()


# Identifying nulls on each one of the columms meta_reviews dataset
null_counts_meta_df = meta_df.select([
    sum(col(c).isNull().cast("int")).alias(c) for c in meta_df.columns])

print("Number of Null valuess per column in  Meta Dataset \n")
null_counts_meta_df.show()
```

```
Number of Null valuess per column in  reviews Dataset

+-------+----+----+------+------+------+------+-------+
|user_id|name|time|rating|  text|  pics|  resp|gmap_id|
+-------+----+----+------+------+------+------+-------+
|      0|   0|   0|     0|223258|500664|477283|      0|
+-------+----+----+------+------+------+------+-------+

Number of Null valuess per column in  Meta Dataset

+----+-------+-------+-----------+--------+---------+--------+----------+-------
-------+-----+-----+----+-----+---------------+---+
|name|address|gmap_id|description|latitude|longitude|category|avg_rating|num_of_
reviews|price|hours|MISC|state|relative_results|url|
+----+-------+-------+-----------+--------+---------+--------+----------+-------
-------+-----+-----+----+-----+---------------+---+
|   0|    168|      0|      11258|       0|        0|      62|         0|
0|11189| 4090|3387| 4712|           1808|  0|
```

```
+----+-------+-------+----------+--------+---------+--------+---------+-------
-------+-----+-----+----+-----+---------------+---+
```

[10]:
```
from pyspark.sql import functions as F

print("Number of Null valuess in  Text column: ")
reviews_df.filter(F.col("text").isNull()).count()

# After the execution, we can see that 'text' column has 223258 NULL values
```

```
Number of Null valuess in  Text column:
```

[10]:
```
223258
```

[11]:
```
# First 10 records which have the text as NULL
reviews_df.filter(F.col("text").isNull()).show(10, truncate=True)
```

```
+--------------------+--------------------+-------------+------+----+----+----+-
------------------+
|             user_id|                name|         time|rating|text|pics|resp|
gmap_id|
+--------------------+--------------------+-------------+------+----+----+----+-
------------------+
|1.169785198939306E20|          Holly Anne|1629595392534|
4|NULL|NULL|NULL|0x56c8992b5dee722…|
|1.097489809617870…|       Emma Forsberg|1629598102068|
5|NULL|NULL|NULL|0x56c8992b5dee722…|
|1.069279219833724…|       sherry miller|1630540491895|
5|NULL|NULL|NULL|0x56c8992b5dee722…|
|1.176554580456088…|            Amy Lieb|1630292802235|
4|NULL|NULL|NULL|0x56c8992b5dee722…|
|1.069274254664423…|Saralan Ruth Hancock|1590877018198|
4|NULL|NULL|NULL|0x56c8e0455225be8…|
|1.021430901643758…|            AK Woman|1513575394394|
5|NULL|NULL|NULL|0x56c8e0455225be8…|
|1.170856429848184…|         John Herman|1567992031983|
5|NULL|NULL|NULL|0x540c2519bcfa6cc…|
|1.142961049420734…|     Brianne Belford|1565031512310|
5|NULL|NULL|NULL|0x540c2519bcfa6cc…|
|1.153189101608130…|  Amelia Hendrickson|1565628571929|
5|NULL|NULL|NULL|0x540c2519bcfa6cc…|
|1.165961600203387E20|       Klaryssa lynn|1519350178408|
4|NULL|NULL|NULL|0x56c89632d94b816…|
+--------------------+--------------------+-------------+------+----+----+----+-
------------------+
only showing top 10 rows
```

[12]:
```
# Now we replace null/None values with 'No review'

reviews_df = reviews_df.withColumn(
    "text",
    F.when(
        (F.col("text").isNull()) | (F.col("text") == F.lit("NULL")), ("No␣
 ↪review")
        )
    .otherwise(F.col("text"))
)
```

[13]:
```
reviews_df.select("text").show(20)
# Column Text sample to check NA values were replaced with No review.
```

```
+--------------------+
|                text|
+--------------------+
|We always stay he…|
|Great campground …|
|We tent camped he…|
|This place is jus…|
|Probably the nice…|
|Great, slept like…|
|It is always a tr…|
|Only 3 booths wit…|
|Not a lot going o…|
|       It's a market|
|           No review|
|           No review|
|           No review|
|           No review|
|Great help, and d…|
|Matt was honest a…|
|Max at Happy Camp…|
|Extremely easy to…|
|Always great serv…|
|Probably the best…|
+--------------------+
only showing top 20 rows
```

[14]:
```
# Validation to check that all nulls in column Text were replaced.

replaced_na = reviews_df.filter(F.col("text").isNull()).count()
print(f"Remaining nulls in the 'text' to replace: {replaced_na}")
```

```
Remaining nulls in the 'text' to replace: 0
```

2.1.2 Question 1.1.2

Processing the content in time column

[15]:
```
### The time is on unix time format miliseconds
reviews_df.select("time").show(5)
```

```
+-------------+
|         time|
+-------------+
|1566331951619|
|1504917982385|
|1474765901185|
|1472858535682|
|1529649811341|
+-------------+
only showing top 5 rows
```

[16]:
```
# Converting Unix time format to yyyy-mm-dd in a new colunm 'newtime'
reviews_df = reviews_df.withColumn(
    "newtime",
    F.from_unixtime((F.col("time")/1000).cast("long"), "yyyy-MM-dd")
)

# Comparison between unix time format and yyyy-mm-dd format
reviews_df.select("time", "newtime").show(5, truncate=False)
```

```
+-------------+----------+
|time         |newtime   |
+-------------+----------+
|1566331951619|2019-08-20|
|1504917982385|2017-09-09|
|1474765901185|2016-09-25|
|1472858535682|2016-09-02|
|1529649811341|2018-06-22|
+-------------+----------+
only showing top 5 rows
```

As we can see in the output, the time is stored as Unix timestamp in miliseconds. Hence, that needs to be first ceonverted to seconds and then format.

[17]:
```
reviews_df.show(5)
```

```
+--------------------+----------------+-------------+------+------------------
-+----+----+--------------------+----------+
|             user_id|            name|         time|rating|
text|pics|resp|             gmap_id|   newtime|
```

```
+--------------------+----------------+-------------+------+--------------------
-+----+----+--------------------+----------+
|1.091298048426862E20|       Nicki Gore|1566331951619|     5|We always stay
he…|NULL|NULL|0x56b646ed2220b77…|2019-08-20|
|1.132409264057589…|    Allen Ratliff|1504917982385|     5|Great campground
…|NULL|NULL|0x56b646ed2220b77…|2017-09-09|
|1.130448378911412…|Jonathan Tringali|1474765901185|     4|We tent camped
he…|NULL|NULL|0x56b646ed2220b77…|2016-09-25|
|1.103291551475920…|            S Blad|1472858535682|     4|This place is
jus…|NULL|NULL|0x56b646ed2220b77…|2016-09-02|
| 1.08989634908602E20|   Daniel Formoso|1529649811341|     5|Probably the
nice…|NULL|NULL|0x56b646ed2220b77…|2018-06-22|
+--------------------+----------------+-------------+------+--------------------
-+----+----+--------------------+----------+
only showing top 5 rows
```

2.2 Question 1.2

2.2.1 Question 1.2.1

Using pyspark to calculate the number of reviews per each unique gmap_id

[18]:
```
# Showing a sample of gmap_id
reviews_df.select("gmap_id").show(5)
```

```
+--------------------+
|             gmap_id|
+--------------------+
|0x56b646ed2220b77…|
|0x56b646ed2220b77…|
|0x56b646ed2220b77…|
|0x56b646ed2220b77…|
|0x56b646ed2220b77…|
+--------------------+
only showing top 5 rows
```

[19]:
```
# Checking for null values
reviews_df.filter(F.col("gmap_id").isNull()).count()
```

[19]: 0

There are zero rows with NULL gmap_id in reviews_df. That's good for us as we don't have to modify any records

[27]:
```
# Calculating  number of reviews per each unique gmap_id
reviews_per_gmap = (
    reviews_df
```

```
    .groupBy("gmap_id")
    .agg(F.count("*").cast("float").alias("total_per_gmap_id"))
    .orderBy(F.col("total_per_gmap_id").desc())
)
```

[28]:
```
# Counting reviews per gmap_id and showing top five rows
reviews_per_gmap.show(5, truncate = False)

reviews_per_gmap.printSchema()
```

```
+-----------------------------------+-----------------+
|gmap_id                            |total_per_gmap_id|
+-----------------------------------+-----------------+
|0x56c897b9ce6000df:0xd707e127588a8c6c|2833.0           |
|0x56c899d058920487:0x12882cc9133f2f54|2594.0           |
|0x56c897c63697ee39:0x419904ababbc740b|2258.0           |
|0x56c8965ee2fb87a1:0x559736347bd48842|2237.0           |
|0x56c89629bde7481f:0x7e8a9413ab25d5d |2219.0           |
+-----------------------------------+-----------------+
only showing top 5 rows

root
 |-- gmap_id: string (nullable = true)
 |-- total_per_gmap_id: float (nullable = false)
```

2.2.2 Question 1.2.2

Transforming the current pyspark dataframe to pandas dataframe

[29]:
```
import pandas as pd

# Transforming PySpark dataframe to Pandas dataframe
df = reviews_df.toPandas()
```

[30]:
```
# Converting time which is in milliseconds to a datetime

df["time"] = pd.to_numeric(df["time"], errors="coerce")
df["time"] = pd.to_datetime(df["time"], unit="ms", errors="coerce")

print(df["time"].head())
print(df["time"].dtype)
```

```
0   2019-08-20 20:12:31.619
1   2017-09-09 00:46:22.385
2   2016-09-25 01:11:41.185
3   2016-09-02 23:22:15.682
4   2018-06-22 06:43:31.341
```

```
Name: time, dtype: datetime64[ns]
datetime64[ns]
```

[31]:
```
# review_time column extract hour from datetime
df["review_time"] = df["time"].dt.strftime("%H:00") # extract hour

# Show first 5 rows
df.head(5)
```

[31]:

	user_id	name	time	rating	\
0	1.091298e+20	Nicki Gore	2019-08-20 20:12:31.619	5	
1	1.132409e+20	Allen Ratliff	2017-09-09 00:46:22.385	5	
2	1.130448e+20	Jonathan Tringali	2016-09-25 01:11:41.185	4	
3	1.103292e+20	S Blad	2016-09-02 23:22:15.682	4	
4	1.089896e+20	Daniel Formoso	2018-06-22 06:43:31.341	5	

	text	pics	resp	\
0	We always stay here when in Valdez for silver …	None	None	
1	Great campground for the price. Nice hot unlim…	None	None	
2	We tent camped here for 2 nights while explori…	None	None	
3	This place is just a few miles outside Valdez,…	None	None	
4	Probably the nicest and cleanest campground we…	None	None	

	gmap_id	newtime	review_time
0	0x56b646ed2220b77f:0xd8975e316de80952	2019-08-20	20:00
1	0x56b646ed2220b77f:0xd8975e316de80952	2017-09-09	00:00
2	0x56b646ed2220b77f:0xd8975e316de80952	2016-09-25	01:00
3	0x56b646ed2220b77f:0xd8975e316de80952	2016-09-02	23:00
4	0x56b646ed2220b77f:0xd8975e316de80952	2018-06-22	06:00

Printing pandas dataframe with top 5 rows after creating the column review_time which now is showing absolute time values.

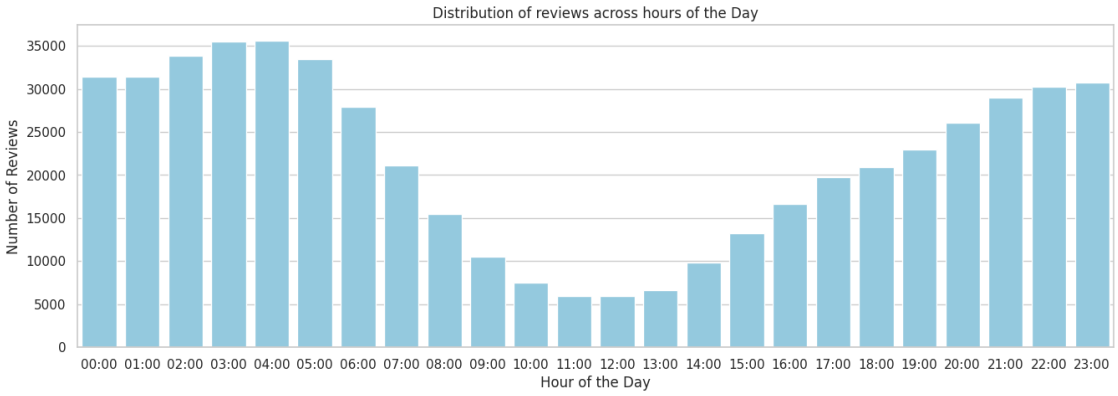
[33]:
```
df['review_time'] # hour level information
```

[33]:
```
0         20:00
1         00:00
2         01:00
3         23:00
4         06:00
          …
521510    00:00
521511    01:00
521512    04:00
521513    19:00
521514    15:00
Name: review_time, Length: 521515, dtype: object
```

2.2.3 Question 1.2.3

Using matplotlib for visualizations on the relationship between gmap_id and reivew_time.

[34]:
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style = 'whitegrid')

# This part creates an ordered list of all hours formatted as 00:00-23:00
hour_order = [f"{h:02d}:00" for h in range(24)]

# Distribution of reviews across hours of day
plt.figure(figsize = (16, 5))

sns.countplot(x = 'review_time', data = df, color = 'skyblue',order=hour_order)
plt.title("Distribution of reviews across hours of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Reviews")
plt.show()
```



Review distribution over the day

The barchart shows that most reviews are concentrated during late night and early morning hours (0–6h).

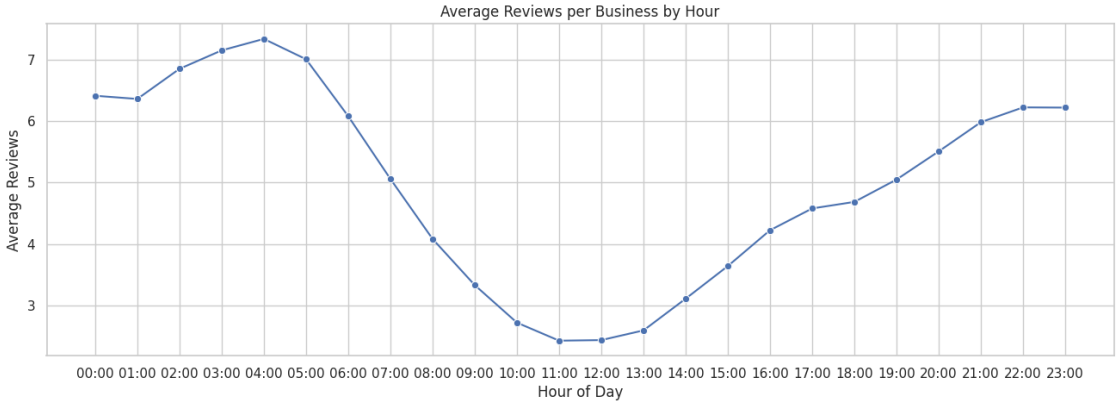
After 7 AM, there is a decline, with the lowest activity between 10 AM and 1 PM.

Reviews increase again from the afternoon (16h), peaking in the evening (20–23h).

Very few reviews are posted around midday, suggesting that people are less likely to leave reviews during work or active business hours.

[35]:
```
# Reviews per business at different times
```

```
reviews_by_time = df.groupby(["review_time", "gmap_id"]).size().
 ↪reset_index(name="count")
avg_reviews = reviews_by_time.groupby("review_time")["count"].mean().
 ↪reset_index()

plt.figure(figsize=(16,5))
sns.lineplot(x="review_time", y="count", data=avg_reviews, marker="o")
plt.title("Average Reviews per Business by Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Average Reviews")
plt.show()
```



Average reviews per business by hour

The second chart shows that businesses receive more reviews late at night (2–5h), where the average is over 7 reviews per business.

The lowest averages is around 10–12h, where businesses average fewer than 3 reviews per business.

Evening hours (18–22h) again show a rise in review activity, aligning with when customers finish shopping, dining, or other services.

2.3 Question 1.3

2.3.1 Question 1.3.1

Determining which workday (day of the week), generates the most reviews

[36]:
```
# Firstly, converting the meta_df to Pandas
meta_pd = meta_df.toPandas()

# checking colunm names for both datasets
print(list(meta_pd.columns))
```

```
['name', 'address', 'gmap_id', 'description', 'latitude', 'longitude',
'category', 'avg_rating', 'num_of_reviews', 'price', 'hours', 'MISC', 'state',
'relative_results', 'url']
```

[37]:
```
print(list(df.columns))
```

```
['user_id', 'name', 'time', 'rating', 'text', 'pics', 'resp', 'gmap_id',
'newtime', 'review_time']
```

[38]:
```
# Before joining the dataset, it is necessary to rename the column 'name' to␣
 ↪avoid errors.
df_renamed= df.rename(columns={'name': 'reviewer_name'})
meta_pd= meta_pd.rename(columns={'name': 'business_name'})

# Now that the reviews dataset and the metadata are aligned, we've
# joined them using the gmap_id column, which serves as the common key between␣
 ↪both datasets.

merged_df = pd.merge(df_renamed, meta_pd, on='gmap_id', how='left')
```

[39]:
```
merged_df.columns
```

[39]:
```
Index(['user_id', 'reviewer_name', 'time', 'rating', 'text', 'pics', 'resp',
       'gmap_id', 'newtime', 'review_time', 'business_name', 'address',
       'description', 'latitude', 'longitude', 'category', 'avg_rating',
       'num_of_reviews', 'price', 'hours', 'MISC', 'state', 'relative_results',
       'url'],
      dtype='object')
```

[40]:
```
# A new column has been added to extract the weekday corresponding to each␣
 ↪business rating.
merged_df['Weekday'] = merged_df['time'].dt.day_name()

merged_df.columns
```

[40]:
```
Index(['user_id', 'reviewer_name', 'time', 'rating', 'text', 'pics', 'resp',
       'gmap_id', 'newtime', 'review_time', 'business_name', 'address',
       'description', 'latitude', 'longitude', 'category', 'avg_rating',
       'num_of_reviews', 'price', 'hours', 'MISC', 'state', 'relative_results',
       'url', 'Weekday'],
      dtype='object')
```

[41]:
```
# Here we use group by to ideentify how man reviews per day  Grouping by Weekday
reviews_by_day = (
    merged_df.groupby('Weekday')
    .size()
    .reset_index(name = 'review_count')
```

```
)

reviews_by_day
```

[41]:

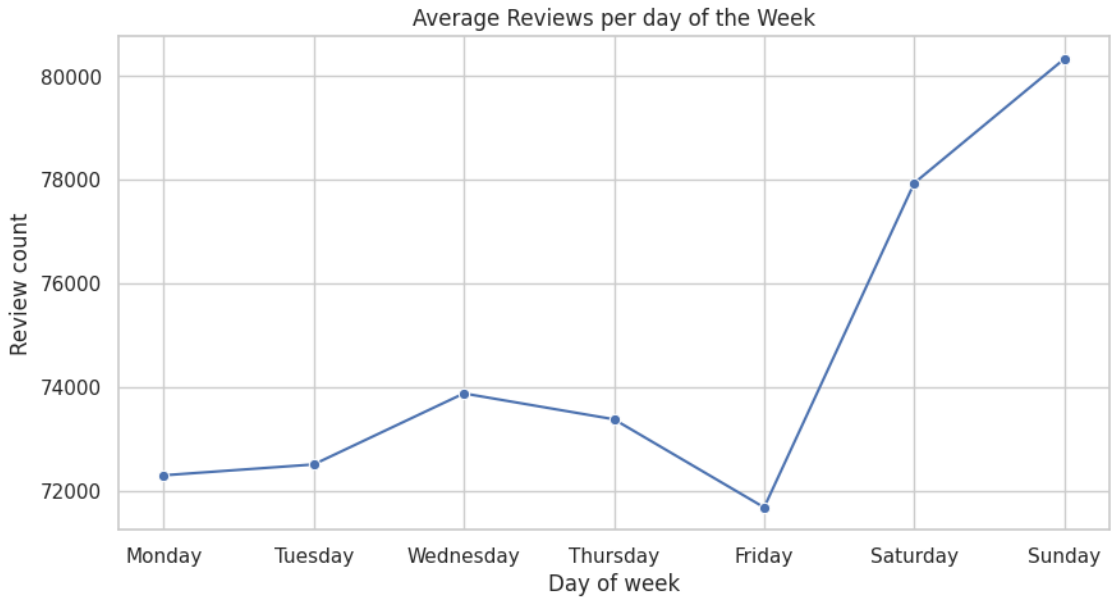
	Weekday	review_count
0	Friday	71673
1	Monday	72293
2	Saturday	77939
3	Sunday	80339
4	Thursday	73375
5	Tuesday	72503
6	Wednesday	73875

[42]:
```
# Ordering the reviews_by_day by real days of the week

weekday_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",␣
 ↪"Saturday", "Sunday"]
reviews_by_day['Weekday'] = pd.Categorical(reviews_by_day['Weekday'],␣
 ↪categories = weekday_order)
reviews_by_day = reviews_by_day.sort_values('Weekday')

reviews_by_day
```

[42]:

	Weekday	review_count
1	Monday	72293
5	Tuesday	72503
6	Wednesday	73875
4	Thursday	73375
0	Friday	71673
2	Saturday	77939
3	Sunday	80339

[43]:
```
# Plotting the line plot for the days of the week on the X-axis and number of
# reviews on the Y axis

plt.figure(figsize = (10, 5))
sns.lineplot(x = "Weekday", y = 'review_count', data = reviews_by_day, marker =␣
 ↪'o')
plt.title("Average Reviews per day of the Week")
plt.xlabel("Day of week")
plt.ylabel("Review count")
plt.grid(True)
plt.show()
```



The line graph shows that people are significantly more active with reviews on the weekend—Sunday tops the chart with over 80,000 reviews, followed closely by Saturday. During the week, Wednesday and Thursday have the most action, while Monday, Tuesday, and especially Friday lag are behind.

2.3.2 Question 1.3.2

Identify the name of the business that has the highest average ratings on that workday

[44]:
```
# First, printint eh best week day
best_day = reviews_by_day.loc[reviews_by_day['review_count'].idxmax(),␣
 ↪'Weekday']
best_day
```

[44]:
```
'Sunday'
```

[45]:
```
# Filter reviews for top day
df_best_day = merged_df[merged_df['Weekday'] == best_day]

df_best_day.head()
```

[45]:
```
          user_id      reviewer_name                       time  rating  \
4   1.130448e+20  Jonathan Tringali 2016-09-25 01:11:41.185       4
5   1.130448e+20  Jonathan Tringali 2016-09-25 01:11:41.185       4
12  1.163932e+20             Matt H 2021-07-04 03:27:50.215       5
13  1.163932e+20             Matt H 2021-07-04 03:27:50.215       5
20  1.169785e+20         Holly Anne 2021-08-22 01:23:12.534       4
```

```
                                                  text  \
4   We tent camped here for 2 nights while explori…
5   We tent camped here for 2 nights while explori…
12  It is always a treat to visit this rain or shi…
13  It is always a treat to visit this rain or shi…
20                                          No review

                                                  pics  resp  \
4                                                 None  None
5                                                 None  None
12  [{'url': ['https://lh5.googleusercontent.com/p…  None
13  [{'url': ['https://lh5.googleusercontent.com/p…  None
20                                                None  None

                                  gmap_id     newtime review_time  …  \
4   0x56b646ed2220b77f:0xd8975e316de80952  2016-09-25       01:00  …
5   0x56b646ed2220b77f:0xd8975e316de80952  2016-09-25       01:00  …
12  0x56c8992b5dee7225:0x9f7f4bf151868cf7  2021-07-04       03:00  …
13  0x56c8992b5dee7225:0x9f7f4bf151868cf7  2021-07-04       03:00  …
20  0x56c8992b5dee7225:0x9f7f4bf151868cf7  2021-08-22       01:00  …

                                          category avg_rating  \
4   ['RV park', 'Cabin rental agency', 'Campground']        4.5
5   ['RV park', 'Cabin rental agency', 'Campground']        4.5
12                             ["Farmers' market"]        4.2
13                             ["Farmers' market"]        4.2
20                             ["Farmers' market"]        4.2

   num_of_reviews  price                                                hours  \
4              18   None                                                 None
5              18   None                                                 None
12             18   None  [['Thursday', 'Closed'], ['Friday', '10AM-5PM'…
13             18   None  [['Thursday', 'Closed'], ['Friday', '10AM-5PM'…
20             18   None  [['Thursday', 'Closed'], ['Friday', '10AM-5PM'…

                                                  MISC  \
4                                                 None
5                                                 None
12  {'Service options': ['In-store shopping'], 'Ac…
13  {'Service options': ['In-store shopping'], 'Ac…
20  {'Service options': ['In-store shopping'], 'Ac…

                    state  \
4                    None
5                    None
12  Closed   Opens 10AM Fri
13  Closed   Opens 10AM Fri
```

```
20  Closed   Opens 10AM Fri

                                      relative_results  \
4   ['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…
5   ['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…
12                                                 None
13                                                 None
20                                                 None

                                                   url Weekday
4   https://www.google.com/maps/place//data=!4m2!3…  Sunday
5   https://www.google.com/maps/place//data=!4m2!3…  Sunday
12  https://www.google.com/maps/place//data=!4m2!3…  Sunday
13  https://www.google.com/maps/place//data=!4m2!3…  Sunday
20  https://www.google.com/maps/place//data=!4m2!3…  Sunday

[5 rows x 25 columns]
```

[47]:
```python
# Average rating per business (gmap_id)
avg_rating_per_bitz = (
    df_best_day.groupby(["gmap_id", "business_name", "category"])
    .agg(
        average_rating = ("rating", "mean"),
        review_count = ("rating", "count")
    )
)

avg_rating_per_bitz = avg_rating_per_bitz.sort_values(
    by=["average_rating", "review_count"],
    ascending=[False, False]
).reset_index()

'''The operation groupby(['gmap_id', 'business_name', 'category']) tells pandas
to identify unique combinations of these three fields.
For each unique group of (gmap_id, business_name, category), it collects all
corresponding rows together.
The .mean() function then computes the average of the rating column for each
group. This summarizes ratings at the granularity defined by the three keys.'''

avg_rating_per_bitz.head(10)
```

[47]:
```
                                gmap_id  \
0  0x56c89869bfea62e3:0xf8778ab6f95058b5
1   0x56c8e8d80099933b:0xe713b933ea9ad13
2   0x56c864be40274793:0xafe08d3ac6f4d5d
3  0x56c897c931af014b:0x7ce8d189db1c8a1b
4  0x56c8e27781137c49:0x1a7d7c5b7d55628d
```

```
5  0x56c663ee84279121:0x37657b5a91a696ec
6  0x56c67c9ee8c294af:0x5ccab006ce0db3f6
7  0x56c80c67ad848299:0x95b9aeb0da99ad5c
8  0x56c896283268dc03:0x25aebb4d6fbb540b
9  0x56c89760c494bc1d:0x6a9dc3c011983c13

                                      business_name  \
0                            Anchorage Ace Hardware
1                                  Escape Anchorage
2  Winner Creek Gorge Trail Head (Lower Winner Cr…
3                           Alaska Veterinary Clinic
4                             Bleeding Heart Brewery
5                                Lucky Raven Tobacco
6                           Kenai Veterinary Hospital
7                     Varly's Swiftwater Seafood Cafe
8  Midnight Market | Vape And Smoke Shop In Ancho…
9                       Alaska Axe Co. - Axe Throwing

                                             category  average_rating  \
0  ['Hardware store', 'Home improvement store', '…             5.0
1              ['Escape room center', 'Entertainer']              5.0
2              ['Hiking area', 'Tourist attraction']              5.0
3           ['Veterinarian', 'Veterinary pharmacy']              5.0
4                                       ['Brewery']              5.0
5                                    ['Tobacco shop']              5.0
6                ['Animal hospital', 'Veterinarian']              5.0
7              ['Seafood restaurant', 'Bar', 'Cafe']              5.0
8  ['Vaporizer store', 'Cigar shop', 'Hookah stor…             5.0
9  ['Sports club', 'Corporate entertainment servi…             5.0

   review_count
0            16
1            16
2            15
3            13
4            13
5            12
6            11
7            11
8            11
9            11
```

[62]:
```
best_business = avg_rating_per_bitz.head(1)   # take only the top 1

best_business
```

[62]:
```
                                  gmap_id           business_name  \
0  0x56c89869bfea62e3:0xf8778ab6f95058b5  Anchorage Ace Hardware

                                              category  average_rating  \
0  ['Hardware store', 'Home improvement store', '…             5.0

   review_count
0            16
```

Since multiple businesses had an average rating of 5, it was necessary also consider the total number of reviews. However, for a more accurate comparison, it may be important to strike a balance between the number of reviews and the final score, as a higher volume of reviews generally provides a more reliable measure of quality.

2.3.3 Question 1.3.3

Finding out insights such as which category it is and what are the peak hours

[65]:
```
best_business_nm = avg_rating_per_bitz.iloc[0]["business_name"]
top_category = merged_df.loc[merged_df["business_name"] == best_business_nm,␣
 ↪"category"].mode().iat[0]

top_category
```

[65]:
```
"['Hardware store', 'Home improvement store', 'Paint store', 'Tool store']"
```

[67]:
```
top_cat_df = merged_df[merged_df['category'] == top_category].copy()

print("Top category:", top_category)
```

```
Top category: ['Hardware store', 'Home improvement store', 'Paint store', 'Tool
store']
```

[68]:
```
top_cat_df.head()
```

[68]:
```
            user_id                reviewer_name                       time  \
18555  1.082242e+20                  trish swain 2021-06-14 02:49:11.869
18556  1.003259e+20                Carolynn Welte 2021-04-23 05:35:28.638
18557  1.004787e+20           Kim n kurt Denning 2020-11-24 20:00:51.065
18558  1.074634e+20                Robert Wisdom 2020-12-05 16:25:04.197
18559  1.068451e+20  Our Crazy Alaskan Adventure 2021-05-27 17:47:18.578

       rating                                               text  pics  resp  \
18555       1  People don't know what they have or are helpfu…  None  None
18556       5  Super friendly employees that always go above …  None  None
18557       5  Absolutely love having a hardware store in Big…  None  None
18558       5  They have great employees,there willing to hel…  None  None
18559       5                              Always helpful staff!  None  None
```

```
                                     gmap_id     newtime review_time  …  \
18555  0x56c8db6333dc407d:0xbcfa2a8075c2981c  2021-06-14       02:00  …
18556  0x56c8db6333dc407d:0xbcfa2a8075c2981c  2021-04-23       05:00  …
18557  0x56c8db6333dc407d:0xbcfa2a8075c2981c  2020-11-24       20:00  …
18558  0x56c8db6333dc407d:0xbcfa2a8075c2981c  2020-12-05       16:00  …
18559  0x56c8db6333dc407d:0xbcfa2a8075c2981c  2021-05-27       17:00  …

                                              category avg_rating  \
18555  ['Hardware store', 'Home improvement store', '…        4.3
18556  ['Hardware store', 'Home improvement store', '…        4.3
18557  ['Hardware store', 'Home improvement store', '…        4.3
18558  ['Hardware store', 'Home improvement store', '…        4.3
18559  ['Hardware store', 'Home improvement store', '…        4.3

      num_of_reviews  price  \
18555             26   None
18556             26   None
18557             26   None
18558             26   None
18559             26   None

                                                hours  \
18555  [['Saturday', '8AM-8PM'], ['Sunday', '8AM-7PM'…
18556  [['Saturday', '8AM-8PM'], ['Sunday', '8AM-7PM'…
18557  [['Saturday', '8AM-8PM'], ['Sunday', '8AM-7PM'…
18558  [['Saturday', '8AM-8PM'], ['Sunday', '8AM-7PM'…
18559  [['Saturday', '8AM-8PM'], ['Sunday', '8AM-7PM'…

                                                  MISC              state  \
18555  {'Service options': ['In-store shopping', 'Del…  Open  Closes 8PM
18556  {'Service options': ['In-store shopping', 'Del…  Open  Closes 8PM
18557  {'Service options': ['In-store shopping', 'Del…  Open  Closes 8PM
18558  {'Service options': ['In-store shopping', 'Del…  Open  Closes 8PM
18559  {'Service options': ['In-store shopping', 'Del…  Open  Closes 8PM

       relative_results                                                 url  \
18555              None  https://www.google.com/maps/place//data=!4m2!3…
18556              None  https://www.google.com/maps/place//data=!4m2!3…
18557              None  https://www.google.com/maps/place//data=!4m2!3…
18558              None  https://www.google.com/maps/place//data=!4m2!3…
18559              None  https://www.google.com/maps/place//data=!4m2!3…

        Weekday
18555    Monday
18556    Friday
18557   Tuesday
```

```
18558  Saturday
18559  Thursday

[5 rows x 25 columns]
```

[69]:
```
top_cat_df.columns
```

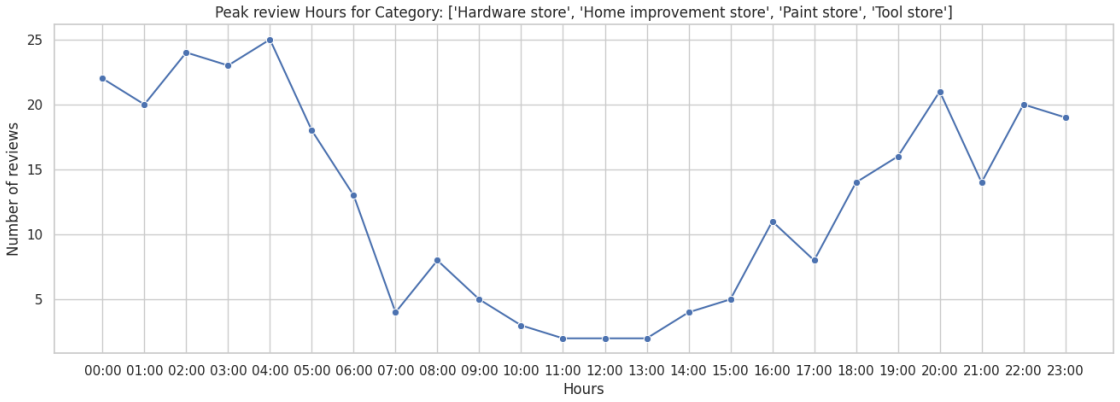
[69]:
```
Index(['user_id', 'reviewer_name', 'time', 'rating', 'text', 'pics', 'resp',
       'gmap_id', 'newtime', 'review_time', 'business_name', 'address',
       'description', 'latitude', 'longitude', 'category', 'avg_rating',
       'num_of_reviews', 'price', 'hours', 'MISC', 'state', 'relative_results',
       'url', 'Weekday'],
      dtype='object')
```

[70]:
```
reviews_by_hour = (
    top_cat_df.groupby('review_time')
    .size()
    .reset_index(name = 'review_count')
)

reviews_by_hour
```

[70]:

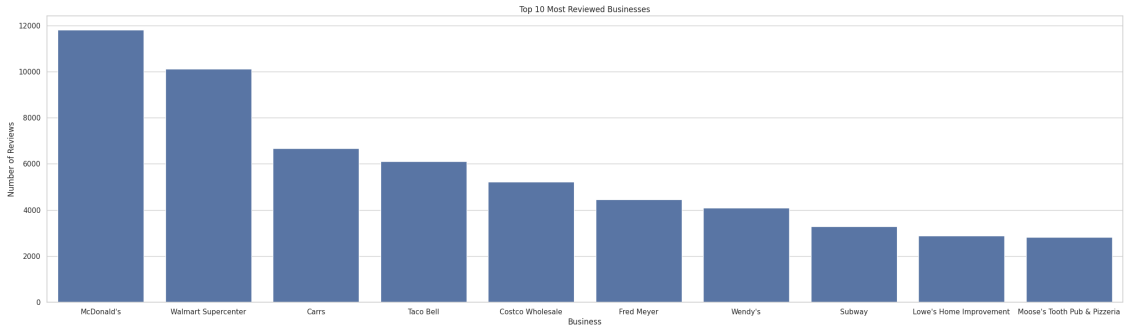
	review_time	review_count
0	00:00	22
1	01:00	20
2	02:00	24
3	03:00	23
4	04:00	25
5	05:00	18
6	06:00	13
7	07:00	4
8	08:00	8
9	09:00	5
10	10:00	3
11	11:00	2
12	12:00	2
13	13:00	2
14	14:00	4
15	15:00	5
16	16:00	11
17	17:00	8
18	18:00	14
19	19:00	16
20	20:00	21
21	21:00	14
22	22:00	20
23	23:00	19

[71]:
```
plt.figure(figsize = (16, 5))
sns.lineplot(x = 'review_time',
             y = 'review_count',
             data = reviews_by_hour,
             marker = 'o')
plt.title(f"Peak review Hours for Category: {top_category}")
plt.xlabel("Hours")
plt.ylabel("Number of reviews")
plt.grid(True)
plt.show()
```



Hence, from the lineplot, we can see that the peak hour for review is around 4 a.m and the number of reviews for category [‘Hardware store’, ‘Home improvement store’, ‘Paint store’, ‘Tool store’] (which is the top category) is more than 25.

[72]:
```
# Top 10 most reviewed businesses
reviews_by_biz = (
    merged_df.groupby(['business_name', 'category'])
    .size()
    .reset_index(name = 'review_count')
    .sort_values('review_count', ascending = False)
    .head(10)
)

plt.figure(figsize = (30, 8))
sns.barplot(data = reviews_by_biz,
            x = 'business_name',
            y = 'review_count'
            )
plt.title("Top 10 Most Reviewed Businesses")
plt.xlabel("Business")
plt.ylabel("Number of Reviews")
plt.show()
```



The bar graph shows that McDonald's has the highest number of reviews overall. However, to determine which business is actually the best, further data analysis is needed—this graph doesn't distinguish between positive and negative reviews, so volume alone doesn't tell the full story.

[73]:
```
import numpy as np

'''A more accurate way to represent reviews per business would be using
stacked bar charts that show the percentage breakdown of each rating level.'''

merged_df_temp = merged_df.copy()
merged_df_temp['rating'] = merged_df_temp['rating'].apply(np.ceil).astype(int)
# First, let's get the rating distribution for each top business

rating_counts = (
    merged_df_temp[merged_df_temp['business_name'].
 ↪isin(reviews_by_biz['business_name'])]
    .groupby(['business_name', 'rating'])
    .size()
    .unstack(fill_value=0)
)

rating_colors = { 1: '#ff6b6b', 2: '#ffa500', 3: '#ffd93d', 4: '#6bcf7f', 5:␣
 ↪'#4ecdc4'}
# Red for 1 star, Orange for 2 stars, Yellow for 3 stars, Light green for 4␣
 ↪stars, Light blue for 5 stars

# Total reviews per business
total_reviews = rating_counts.sum(axis=1)

rating_pct = rating_counts.div(total_reviews, axis=0) * 100

# necessary to order businesses by total reviews
rating_counts = rating_counts.loc[total_reviews.sort_values(ascending=False).
 ↪index]
rating_pct = rating_pct.loc[rating_counts.index]
```

```
# Plot each rating category as a stacked bar
plt.figure(figsize=(25, 8))
bottom = np.zeros(len(rating_counts))

for rating in sorted(rating_counts.columns, reverse=True):
    plt.bar(
        rating_counts.index,
        rating_counts[rating],
        bottom=bottom,
        color=rating_colors[rating],
        label=f'{rating} Star'
    )
    bottom += rating_counts[rating]

plt.title("Top Businesses - Ratings & Review Counts", fontsize=16,␣
 ↪fontweight='bold')
plt.xlabel("Business", fontsize=12)
plt.ylabel("Number of Reviews", fontsize=12)
plt.xticks(rotation=90, ha='right')
plt.legend(title='Rating', bbox_to_anchor=(1.05, 1), loc='upper left')


for i, biz in enumerate(rating_counts.index):
    cum = 0
    for rating in sorted(rating_counts.columns, reverse=True):
        count = rating_counts.loc[biz, rating]
        if count > 200:
            plt.text(
                i,
                cum + count / 2,
                f"{rating_pct.loc[biz, rating]:.1f}%",
                ha="center", va="center",
                fontsize=8, fontweight="bold", color="gray"
            )
        cum += count
```

Creating a macro category calsification for businesess

Previously, we found that "['Hardware store', 'Home improvement store', 'Paint store', 'Tool store']" was the category with the highest number of reviews. However, this classification may be biased, as the category column includes multiple subcategories and overlaps that could distort the representation. To address this, we introduced a set of macro categories that group related subcategories under broader labels, allowing for a more balanced and meaningful comparison across business types.

[75]:
```
import re
from collections import Counter
from wordcloud import STOPWORDS

# Ensure everything is a string, then join all category values
all_words = " ".join(merged_df["category"].fillna("").astype(str))

# Remove punctuation and lowercase everything
clean_text = re.sub(r"[^A-Za-z\s]", " ", all_words).lower()

# Split into tokens and remove stopwords/short words
tokens1 = [w for w in clean_text.split() if w not in STOPWORDS and len(w) > 2]

# Count frequencies
common_words = Counter(tokens1).most_common(50)
for word, freq in common_words:
    print(f"{word:20} {freq}")
```

```
restaurant           515241
store                381064
shop                 125975
bar                  61180
service              51958
```

food	49586
american	48872
grocery	47475
fast	45337
tourist	41971
attraction	40428
breakfast	34364
pizza	34048
coffee	33426
hamburger	32194
goods	31982
park	31776
takeout	31623
clothing	31420
car	27446
sandwich	25335
delivery	24656
center	22860
dealer	22330
home	21256
mexican	20374
department	19595
sporting	18994
auto	18798
hotel	18415
cafe	18171
supply	17694
seafood	17499
supermarket	16820
repair	16781
grill	16729
supplier	15830
shopping	15596
mall	15580
asian	14054
craft	13828
toy	13791
electronics	13743
sports	13469
house	13378
furniture	13250
family	12943
venue	12870
steak	12036
pet	11704

[76]:
```
'''We improved category classification by creating macro categories derived␣
 ↪from the most
frequent words in the category column. '''

from itertools import chain

# Define each macro category only once
macro_keywords = {
    "Food & Drink":
     ["restaurant", "bar", "cafe", "coffee", "brewpub", "brewery", "pub",␣
 ↪"grill", "bakery",
      "pizza", "fast", "hamburger", "sandwich", "takeout", "delivery",␣
 ↪"seafood", "chicken",
      "cream", "lunch", "sushi", "steak", "buffet", "breakfast", "asian",␣
 ↪"chinese", "italian",
      "japanese", "mex", "tex", "burrito", "taco", "vegetarian", "dessert",␣
 ↪"wine", "cocktail",
      "barbecue", "wings", "brunch", "ice", "espresso"],

    "Retail & Shopping":
     ["store", "shop", "shopping", "goods", "gift",
      "electronics", "sports", "video","book", "hardware", "appliance",␣
 ↪"phone", "cell",
      "liquor","supermarket", "grocery", "market", "discount", "yarn","supply",␣
 ↪"supplier",
      "department", "warehouse", "materials", "equipment", "lighting",␣
 ↪"improvement", "frame",
      "picture","furniture", "rug", "paint", "mall", "outlet","clothing",␣
 ↪"shoe","toy"],

    "Vehicle Services":
     ["car", "auto", "dealer", "truck", "station", "motorcycle", "brake",␣
 ↪"battery",
      "repair", "oil", "wash", "tire", "wheel", "change", "trailer", "parts",␣
 ↪"propane", "gas"],

    "Lodging & Travel":
     ["hotel", "motel", "resort", "lodge", "campground", "hall","banquet",␣
 ↪"airport", "visitor",
      "tour", "tourist","organization", "agency", "rental"],

    "Parks & Outdoor Recreation":
     ["park", "forest", "hiking", "campground", "outdoor", "hunting",␣
 ↪"fishing", "wildlife",
      "ski", "recreation", "area", "field", "playground", "garden", "lawn",␣
 ↪"complex"],
```

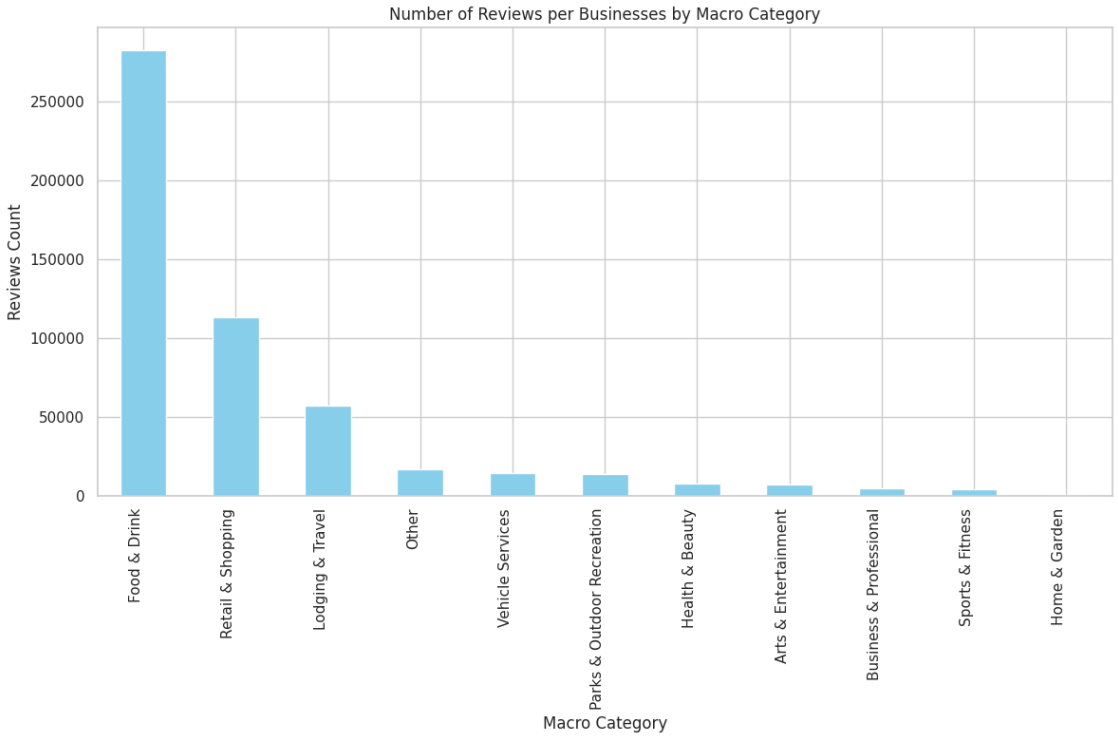
```
    "Health & Beauty":
     ["salon", "spa", "beauty", "body", "health", "treatment","veterinarian",␣
 ↪"hair", "trainer",
      "physical", "therapy"],

    "Sports & Fitness": ["gym", "fitness", "sporting", "sports"],

    "Arts & Entertainment":
     ["museum", "theater", "amusement", "movie", "music", "art", "concert",␣
 ↪"event", "club", "video"],

    "Home & Garden":
     ["home", "house", "garden", "flooring", "window", "building","bathroom",␣
 ↪"cabinet", "hardware",
      "lawn", "improvement","materials", "paint", "clean", "appliance"],

    "Business & Professional":
     ["business", "company", "organization", "provider", "service", "non",␣
 ↪"profit", "agency"],
}

# Build the flat dictionary
macro_map = {
    kw: category
    for category, keywords in macro_keywords.items()
    for kw in keywords
}

# Add fallback
macro_map[""] = "Other"
```

[77]:
```
def classify_category(cat_value):
    """
    cat_value: string or list of strings (e.g., from your 'category' column)
    returns: macro category (first match) or 'Other'
    """
    # Turn list into a single string and lowercase
    if isinstance(cat_value, list):
        text = " ".join(cat_value).lower()
    else:
        text = str(cat_value).lower()

    for keyword, macro in macro_map.items():
        if keyword and keyword in text:
            return macro
    return "Other"
```

[78]:
```
merged_df["macro_category"] = merged_df["category"].apply(classify_category)

merged_df["macro_category"].value_counts()

# Number of reviews by macro category
```

[78]:
```
macro_category
Food & Drink                   282516
Retail & Shopping              112883
Lodging & Travel                57037
Other                           17006
Vehicle Services                14500
Parks & Outdoor Recreation      13709
Health & Beauty                  7976
Arts & Entertainment             6938
Business & Professional          4736
Sports & Fitness                 3955
Home & Garden                     741
Name: count, dtype: int64
```

[79]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,8))
(
    merged_df["macro_category"]
    .value_counts()
    .plot(kind="bar", color="skyblue")
)
plt.title("Number of Reviews per Businesses by Macro Category")
plt.xlabel("Macro Category")
plt.ylabel("Reviews Count")
plt.xticks(rotation=90, ha="right")
plt.tight_layout()
plt.show()
```



To improve the interpretability of business categories, we first extracted the most frequent keywords from the original "category" column. Based on this analysis, we defined a set of representative macro categories such as Food & Drink, Retail & Shopping, and Vehicle Services that capture the dominant themes across the dataset. Each business entry was then mapped to one of these macro categories, and the aggregated counts were visualized using a bar chart.

The results show that Food & Drink is by far the most prevalent category (282,516 entries), followed by Retail & Shopping (112,883) and Lodging & Travel (57,037). Less common categories include Home & Garden (741) and Sports & Fitness (3,955), with a residual "Other" group (17,006) including entries that did not fit into the predefined categories.

2.4 Question 1.4

Review Trends and Word Clouds by Year

[80]:
```
import re
from collections import Counter
from wordcloud import WordCloud, STOPWORDS

merged_df['year'] = merged_df['time'].dt.year # required later for wordcloud
```

[81]:
```
df_text_mining = merged_df.copy()

df_text_mining.head()
```

[81]:

	user_id	reviewer_name	time	rating	\
0	1.091298e+20	Nicki Gore	2019-08-20 20:12:31.619	5	
1	1.091298e+20	Nicki Gore	2019-08-20 20:12:31.619	5	
2	1.132409e+20	Allen Ratliff	2017-09-09 00:46:22.385	5	
3	1.132409e+20	Allen Ratliff	2017-09-09 00:46:22.385	5	
4	1.130448e+20	Jonathan Tringali	2016-09-25 01:11:41.185	4	

	text	pics	resp	\
0	We always stay here when in Valdez for silver …	None	None	
1	We always stay here when in Valdez for silver …	None	None	
2	Great campground for the price. Nice hot unlim…	None	None	
3	Great campground for the price. Nice hot unlim…	None	None	
4	We tent camped here for 2 nights while explori…	None	None	

	gmap_id	newtime	review_time	…	\
0	0x56b646ed2220b77f:0xd8975e316de80952	2019-08-20	20:00	…	
1	0x56b646ed2220b77f:0xd8975e316de80952	2019-08-20	20:00	…	
2	0x56b646ed2220b77f:0xd8975e316de80952	2017-09-09	00:00	…	
3	0x56b646ed2220b77f:0xd8975e316de80952	2017-09-09	00:00	…	
4	0x56b646ed2220b77f:0xd8975e316de80952	2016-09-25	01:00	…	

	num_of_reviews	price	hours	MISC	state	\
0	18	None	None	None	None	
1	18	None	None	None	None	
2	18	None	None	None	None	
3	18	None	None	None	None	
4	18	None	None	None	None	

	relative_results	\
0	['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…	
1	['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…	
2	['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…	
3	['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…	
4	['0x56b6445fd9f9e387:0x6dd3d374ef56431a', '0x5…	

	url	Weekday	\
0	https://www.google.com/maps/place//data=!4m2!3…	Tuesday	
1	https://www.google.com/maps/place//data=!4m2!3…	Tuesday	
2	https://www.google.com/maps/place//data=!4m2!3…	Saturday	
3	https://www.google.com/maps/place//data=!4m2!3…	Saturday	
4	https://www.google.com/maps/place//data=!4m2!3…	Sunday	

	macro_category	year
0	Parks & Outdoor Recreation	2019
1	Parks & Outdoor Recreation	2019
2	Parks & Outdoor Recreation	2017
3	Parks & Outdoor Recreation	2017

```
4  Parks & Outdoor Recreation  2016

[5 rows x 27 columns]
```

[82]:
```
'''The function is used to perfrom text mining. It clears the text and gives a
clear text in return while performing the operaions explained.'''

def clean_text(text):
    """
    Clean and normalise review text:Lowercase, Remove punctuation, digits,␣
 ↪special chars, Remove stopwords
    Collapse multiple spaces
    """
    if not isinstance(text, str):
        return ""

    text = text.lower() # Lowercase
    text = re.sub(r"[^a-z\s]", " ", text) # Keep only letters and whitespace
    text = re.sub(r"\s+", " ", text).strip()# Collapse extra spaces

    tokens = [word for word in text.split() if word not in STOPWORDS] # Remove␣
 ↪stopwords

    return " ".join(tokens)
```

[83]:
```
# Making a new column named 'clean_text' that has the cleaned text
df_text_mining['clean_text'] = df_text_mining['text'].apply(clean_text)

df_text_mining[['text', 'clean_text']].head()
```

[83]:
```
                                                text  \
0  We always stay here when in Valdez for silver …
1  We always stay here when in Valdez for silver …
2  Great campground for the price. Nice hot unlim…
3  Great campground for the price. Nice hot unlim…
4  We tent camped here for 2 nights while explori…

                                          clean_text
0  always stay valdez silver salmon fishing elder…
1  always stay valdez silver salmon fishing elder…
2  great campground price nice hot unlimited show…
3  great campground price nice hot unlimited show…
4  tent camped nights exploring valdez center cam…
```

[84]:
```
print(STOPWORDS)
```

```
{"what's", 'all', 'therefore', "you'd", 'is', 'own', 'k', 'at', 'r', 'hers',
"isn't", 'a', 'nor', 'ever', 'it', "she's", 'between', "we're", "she'll",
"wouldn't", "she'd", 'those', 'ought', 'below', 'www', "why's", 'more', "we'll",
"they're", 'himself', "they've", "we'd", 'against', 'if', 'itself', "shan't",
'i', "here's", "when's", 'until', 'when', 'am', 'hence', 'get', "they'd", 'not',
"i've", 'too', "we've", "he's", 'would', 'both', "where's", 'are', 'or', 'who',
'does', 'what', 'any', "i'm", 'was', 'into', "haven't", 'during', 'had', 'same',
"hadn't", "he'll", 'have', "they'll", 'here', 'my', "wasn't", 'very', 'you',
"hasn't", 'of', 'shall', 'that', 'after', 'an', "let's", "don't", 'with',
'could', 'most', 'again', 'her', 'him', "that's", 'should', 'theirs', 'which',
"mustn't", 'she', 'than', 'herself', 'once', 'he', 'whom', 'other', "how's",
'so', 'there', 'over', 'to', 'off', "won't", 'no', 'because', "he'd", 'be',
'by', 'their', 'above', "doesn't", 'out', 'being', "i'll", 'do', 'myself',
'each', 'then', 'ourselves', "aren't", 'been', "it's", 'we', 'were', "couldn't",
'these', 'themselves', 'as', 'under', 'only', 'further', 'com', 'while', 'some',
'like', 'http', "can't", 'for', 'since', 'from', 'our', 'few', 'and', 'them',
'however', 'yours', 'can', 'down', 'just', 'up', 'else', 'ours', 'doing',
'such', 'me', 'cannot', "you'll", 'your', 'they', 'yourselves', "shouldn't",
'through', 'before', 'his', 'its', 'why', 'has', 'having', 'but', "i'd",
'otherwise', 'yourself', 'the', 'where', 'about', 'how', "didn't", "you've",
"weren't", "there's", 'also', "you're", "who's", 'on', 'in', 'did', 'this'}
```

[85]:
```
print("Text -")
print(df_text_mining['text'].iloc[1])
print("\n")
print("Clean text -")
print(df_text_mining['clean_text'].iloc[1])
```

```
Text -
We always stay here when in Valdez for silver salmon fishing. The elderly couple
that run it are amazing to talk to, extremely helpful. The campsites are very
well maintained.


Clean text -
always stay valdez silver salmon fishing elderly couple run amazing talk
extremely helpful campsites well maintained
```

To prepare the text data for analysis, we applied a preprocessing step that involved converting all characters to lowercase and removing punctuation.

Additionally, we filtered out common stopwords, such as “the,” “and,” “of” which typically do not contribute meaningful information to the classification task. This cleaning process keep core content, allowing for more accurate keyword extraction and category mapping.

[86]:
```
# Finding top 30 frequent qords

# Joining all the words in the clean_text column
# all_words = " ".join(df_text_mining['clean_text'])
```

```
all_words = " ".join(df_text_mining['clean_text'].fillna("").astype(str))

# Splitting into tokens
tokens = [w for w in all_words.split() if w not in STOPWORDS and len(w) > 2]

# Top 30 most frequent words
common_words = Counter(tokens).most_common(30)
for word, freq in common_words:
    print(f"{word:20} {freq}")
```

```
review               224625
great                83356
good                 64716
food                 63822
place                48230
service              46136
staff                29074
friendly             26673
nice                 26645
always               23784
best                 22082
love                 20043
time                 19145
one                  15984
clean                12748
people               12728
really               12686
will                 12645
amazing              12623
prices               12361
store                12242
awesome              12170
back                 12031
well                 11845
helpful              10756
little               10491
excellent            10370
customer             9555
got                  9540
delicious            9480
```

[87]:
```
# Making a dataframe for top 30 common words from the data
common_df = pd.DataFrame(common_words, columns = ['word', 'count'])
common_df
```

[87]:

	word	count
0	review	224625
1	great	83356
2	good	64716
3	food	63822
4	place	48230
5	service	46136
6	staff	29074
7	friendly	26673
8	nice	26645
9	always	23784
10	best	22082
11	love	20043
12	time	19145
13	one	15984
14	clean	12748
15	people	12728
16	really	12686
17	will	12645
18	amazing	12623
19	prices	12361
20	store	12242
21	awesome	12170
22	back	12031
23	well	11845
24	helpful	10756
25	little	10491
26	excellent	10370
27	customer	9555
28	got	9540
29	delicious	9480

[88]:
```
'''Different themes for the wordcloud.
Just put any of these values in the colormap attribute -
["viridis", "plasma", "inferno", "coolwarm", "Set3"]'''

# Word cloud
years = merged_df["year"].unique()

for year in sorted(years):
  year_text = " ".join(
      df_text_mining.loc[df_text_mining['year'] == year, 'clean_text']
      .fillna("")
      .astype(str)
      )
  wc = WordCloud(
      width = 800, height = 400, # size of the image
```

```
    background_color = 'white', # white background
    stopwords = STOPWORDS,
    colormap = 'inferno' # color scheme for words
).generate(year_text)
plt.figure(figsize = (10, 5))
plt.imshow(wc, interpolation = 'bilinear')
plt.axis('off')
plt.title(f"WOrd cloud for reviews in {year}")
plt.show()
print("")
```

WOrd cloud for reviews in 2008

WOrd cloud for reviews in 2009

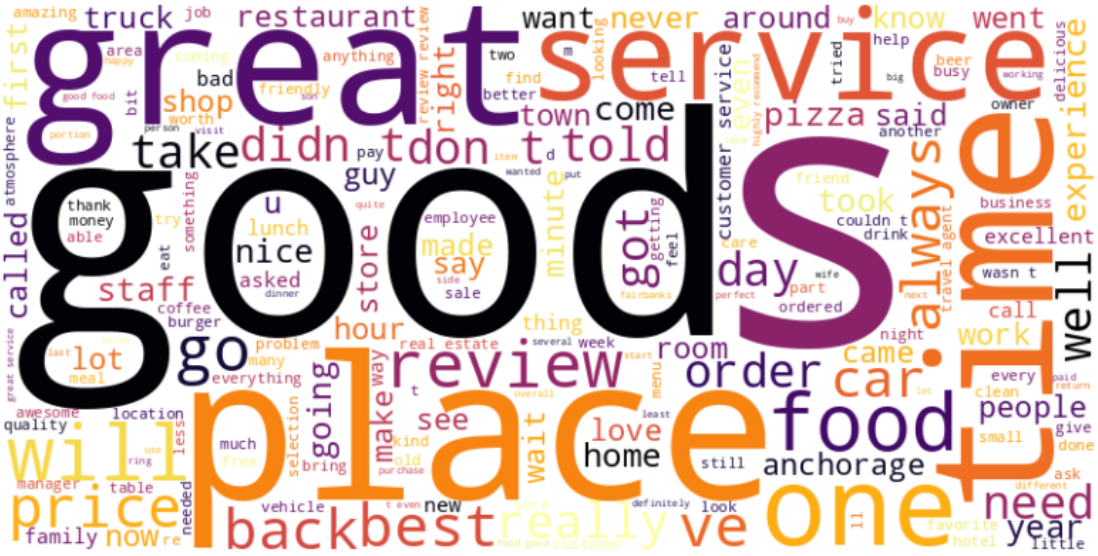
WOrd cloud for reviews in 2010

WOrd cloud for reviews in 2011

WOrd cloud for reviews in 2012

WOrd cloud for reviews in 2013

WOrd cloud for reviews in 2014



WOrd cloud for reviews in 2015

WOrd cloud for reviews in 2016

WOrd cloud for reviews in 2017

WOrd cloud for reviews in 2018

WOrd cloud for reviews in 2019

We performed the NLP tasks in this phase. We performed tokenization by removing the stop words and taking the base form of the word. After this, we took the tokenized base words and found their count in the whole data by combining every row by split(""). Later, we made a word cloud according to the year which gives the most frequent word in that particular year.

2.5 Question 1.5

Analysis on the business_name and the reviewers

[93]:
```
# To find the unique reviewrs, we use the user_id field

# Unique reviewers at the business level
unique_reviewers_business = (
    merged_df.groupby(['gmap_id', 'business_name', 'category'])['user_id']
    .nunique()
    .reset_index(name = 'unique_reviewers')
    .sort_values('unique_reviewers', ascending = False)
)
# Top 10 businesses reviewed by the unique reviewers
unique_reviewers_business.head(10)
```

[93]:

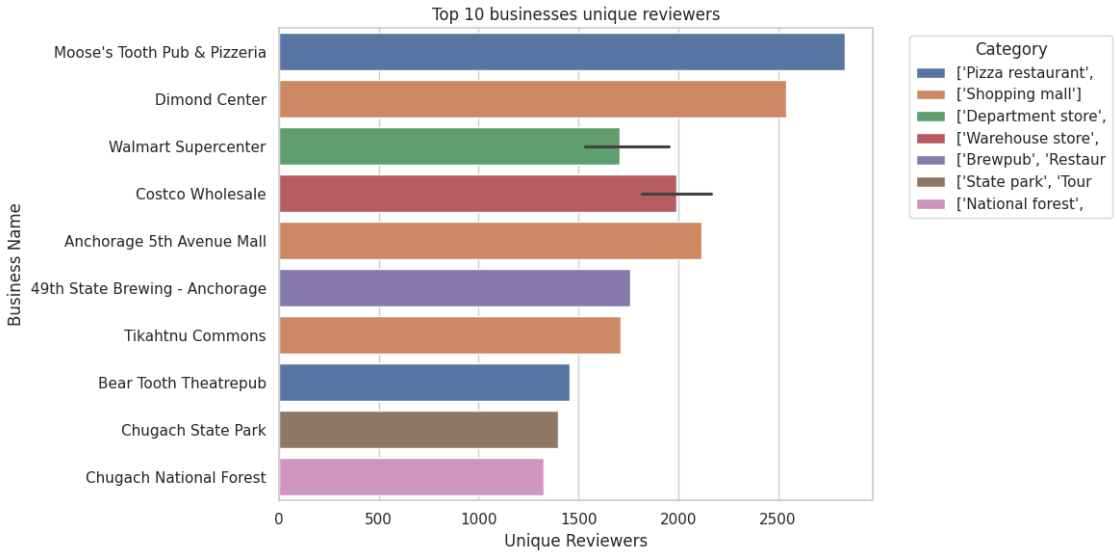
	gmap_id	business_name	category	unique_reviewers
3593	0x56c897b9ce6000df:0xd707e127588a8c6c	Moose's Tooth Pub & Pizzeria	['Pizza restaurant', 'Bar', 'Brewpub', 'Restau…	2833
4370	0x56c899d058920487:0x12882cc9133f2f54	Dimond Center	['Shopping mall']	2542
3730	0x56c897c63697ee39:0x419904ababbc740b	Walmart Supercenter	['Department store', 'Clothing store', 'Craft …	2200
2881	0x56c8965ee2fb87a1:0x559736347bd48842	Costco Wholesale	['Warehouse store', 'Department store']	2164
2653	0x56c89629bde7481f:0x7e8a9413ab25d5d	Anchorage 5th Avenue Mall	['Shopping mall']	2116
4364	0x56c899cd61bbf82b:0x544a5b80b67a93c7	Costco Wholesale	['Warehouse store', 'Department store']	1820
4733	0x56c8bd86fd671871:0x52c896e66d960c02	49th State Brewing - Anchorage	['Brewpub', 'Restaurant']	1761
4368	0x56c899d0184538f9:0xfcf4420cda19d613	Walmart Supercenter	['Department store', 'Clothing store', 'Craft …	1723
2463	0x56c8942639bddcf1:0x3d4151966bed9375	Tikahtnu Commons	['Shopping mall']	1713
3009	0x56c8969817c5b323:0xfd52c71fe0d827f1	Walmart Supercenter	['Department store', 'Clothing store', 'Craft …	1585

[95]:
```
print(len(unique_reviewers_business))
```

```
6012
```

[91]:
```
top10_business_reviewers = unique_reviewers_business.head(15).copy()
top10_business_reviewers["short_category"] =␣
 ↪top10_business_reviewers["category"].str.slice(0, 20)
# It's necessary to shorten the category names to improve plot readability
```

```
plt.figure(figsize = (12, 6))
sns.barplot(data = top10_business_reviewers,
            x = 'unique_reviewers',
            y = 'business_name',
            hue = 'short_category',
            dodge = False
            )
plt.title("Top 10 businesses unique reviewers")
plt.xlabel("Unique Reviewers")
plt.ylabel("Business Name")
plt.legend(title = "Category", bbox_to_anchor=(1.05, 1), loc = "upper left")
plt.tight_layout()
plt.show()
```



As seen from the abr plot, the business "Moose's Tooth Pub & Pizzeria" has got the highest number of unique business reviewers. On the other hand, "Chugach National FOrest" has gpt the lowest number of unique reviewers.

[96]:
```
# unique reviewers per category
unique_reviews_category = (
    merged_df.groupby(['category'])['user_id']
    .nunique()
    .reset_index(name = 'unique_reviewers')
    .sort_values('unique_reviewers', ascending = False)
)

unique_reviews_category.head(5)
```

[96]:

	category	unique_reviewers
2189	['Shopping mall']	7176
866	['Department store', 'Clothing store', 'Craft …	7092
993	['Fast food restaurant', 'Breakfast restaurant…	6951
1206	['Grocery store', 'Grocery delivery service']	5795
1628	['Mexican restaurant']	5062

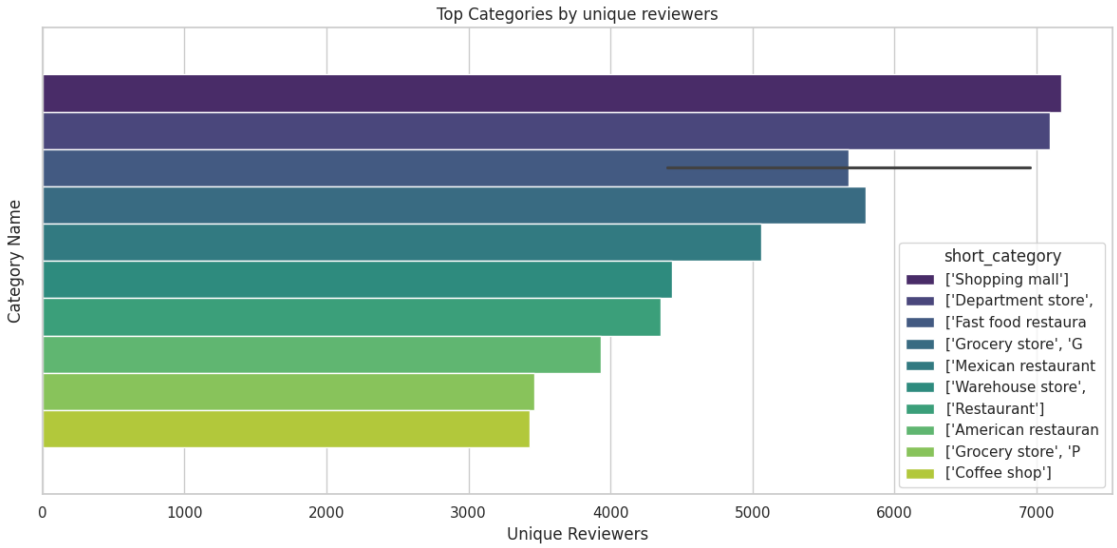
[97]:
```
# unique reviewers per macro_category
unique_reviews_macro_category = (
    merged_df.groupby(['macro_category'])['user_id']
    .nunique()
    .reset_index(name = 'unique_reviewers')
    .sort_values('unique_reviewers', ascending = False)
)

unique_reviews_macro_category.head(10)
```

[97]:

	macro_category	unique_reviewers
2	Food & Drink	19967
8	Retail & Shopping	18036
5	Lodging & Travel	15074
6	Other	9357
10	Vehicle Services	7667
7	Parks & Outdoor Recreation	7298
3	Health & Beauty	5402
0	Arts & Entertainment	5151
1	Business & Professional	3700
9	Sports & Fitness	3133

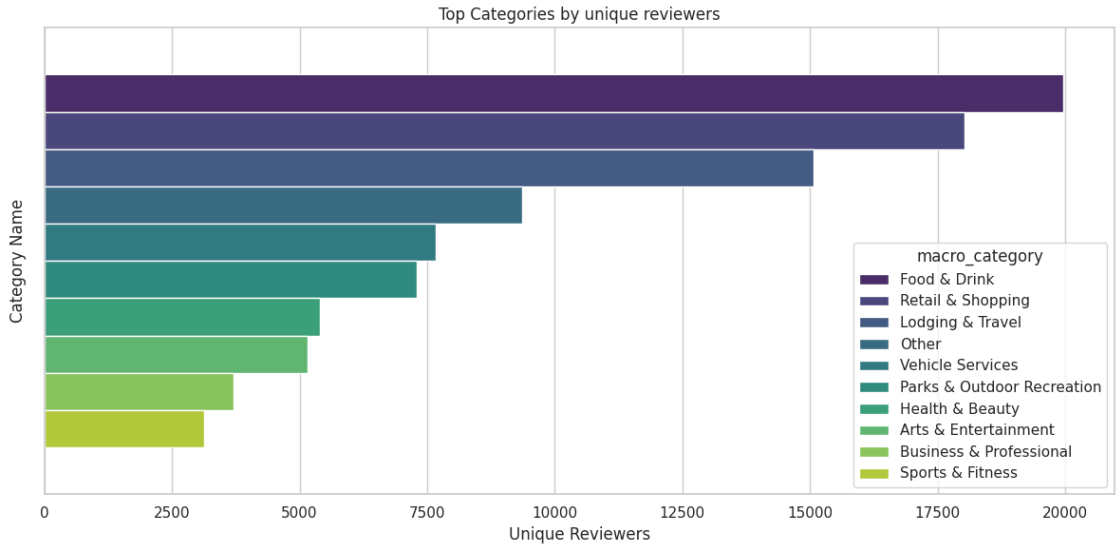
Top10 unique reviews by category

[98]:
```
top10_unique_reviewers_by_category = unique_reviews_category.head(11).copy()
top10_unique_reviewers_by_category["short_category"] =␣
 ↪top10_unique_reviewers_by_category["category"].str.slice(0, 20)

plt.figure(figsize = (12, 6))
sns.barplot(data = top10_unique_reviewers_by_category,
            x = 'unique_reviewers',
            hue = 'short_category',
            palette =  'viridis'
            )
plt.title("Top Categories by unique reviewers")
plt.xlabel("Unique Reviewers")
plt.ylabel("Category Name")
plt.tight_layout()
plt.show()
```



Top10 unique reviews by macro category

[99]:
```
top10_unique_reviews_macro_category = unique_reviews_macro_category.head(10)

plt.figure(figsize = (12, 6))
sns.barplot(data = top10_unique_reviews_macro_category,
            x = 'unique_reviewers',
            hue = 'macro_category',
            palette =  'viridis'
            )
plt.title("Top Categories by unique reviewers")
plt.xlabel("Unique Reviewers")
plt.ylabel("Category Name")
plt.tight_layout()
plt.show()
```



[ ]:
```
merged_df.info()

# To find the temporal patterns by weekday, year, month, hour.
# As seen in the columns, we have the weekday, year and hours columns.
# To find the temporal pattern by month, adding that particular column in the
# merged_df'''
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 521997 entries, 0 to 521996
Data columns (total 27 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   user_id           521997 non-null  float64
 1   reviewer_name     521997 non-null  object
 2   time              521997 non-null  datetime64[ns]
 3   rating            521997 non-null  int32
 4   text              521997 non-null  object
 5   pics              20871 non-null   object
 6   resp              44306 non-null   object
 7   gmap_id           521997 non-null  object
 8   newtime           521997 non-null  object
 9   review_time       521997 non-null  object
 10  business_name     521997 non-null  object
 11  address           521341 non-null  object
 12  description       263503 non-null  object
 13  latitude          521997 non-null  float64
 14  longitude         521997 non-null  float64
 15  category          521616 non-null  object
 16  avg_rating        521997 non-null  float64
```

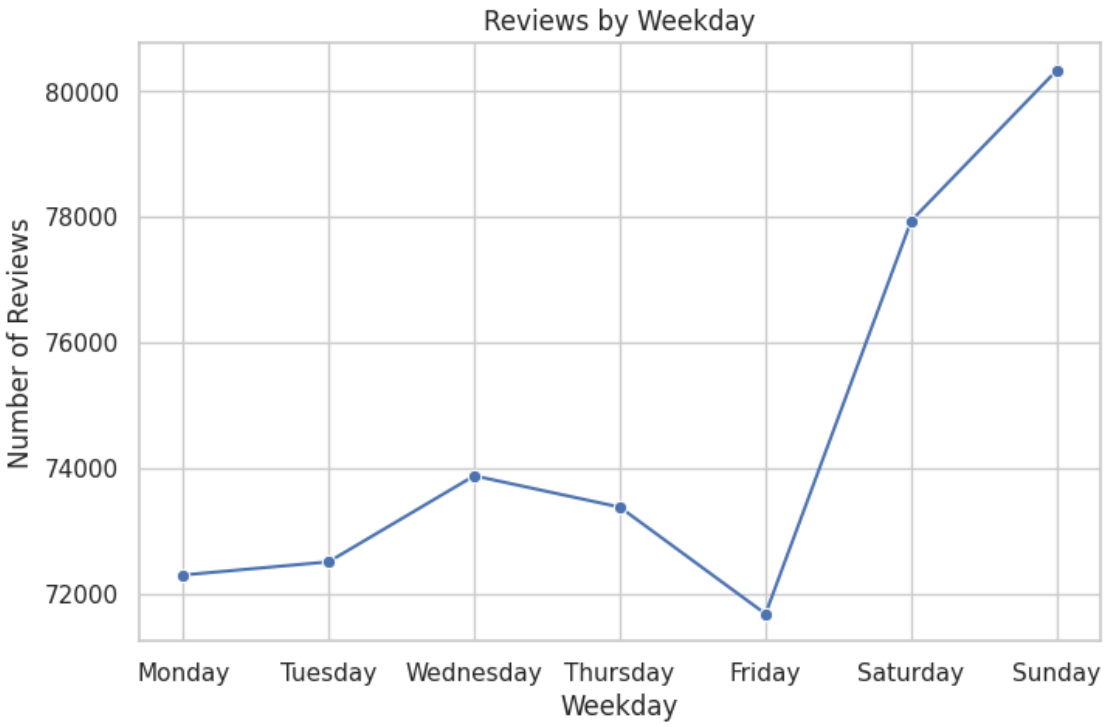
```
 17  num_of_reviews    521997 non-null  int32
 18  price             245753 non-null  object
 19  hours             453059 non-null  object
 20  MISC              486423 non-null  object
 21  state             270876 non-null  object
 22  relative_results  493940 non-null  object
 23  url               521997 non-null  object
 24  Weekday           521997 non-null  object
 25  macro_category    521997 non-null  object
 26  year              521997 non-null  int32
dtypes: datetime64[ns](1), float64(4), int32(3), object(19)
memory usage: 101.6+ MB
```

[101]:
```
merged_df['month'] = merged_df['time'].dt.month_name()

merged_df[['Weekday', 'year', 'month', 'hours']].head()
```

[101]:

	Weekday	year	month	hours
0	Tuesday	2019	August	None
1	Tuesday	2019	August	None
2	Saturday	2017	September	None
3	Saturday	2017	September	None
4	Sunday	2016	September	None

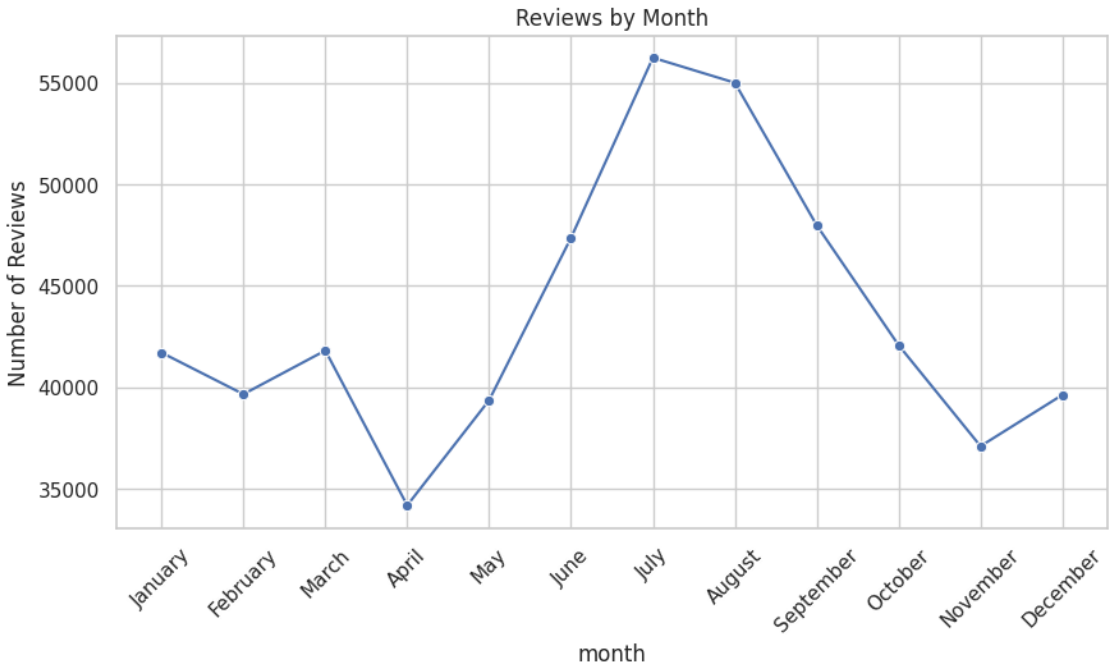
[100]:
```
# Reviews by weekday

reviews_by_day = (
    merged_df.groupby("Weekday")['user_id']
    .count()
    .reset_index(name="review_count")
)

reviews_by_day['Weekday'] = pd.Categorical(reviews_by_day['Weekday'],
                                           categories = weekday_order,␣
 ↪ordered=True)
reviews_by_day = reviews_by_day.sort_values("Weekday")

plt.figure(figsize=(8,5))
sns.lineplot(data=reviews_by_day,
             x = "Weekday",
             y = "review_count",
             marker = "o")
plt.title("Reviews by Weekday")
plt.ylabel("Number of Reviews")
plt.show()
```



There are more than 80000 reviews on SUnday, which is the highest; while, there are less than 72000 reviews on a Friday.

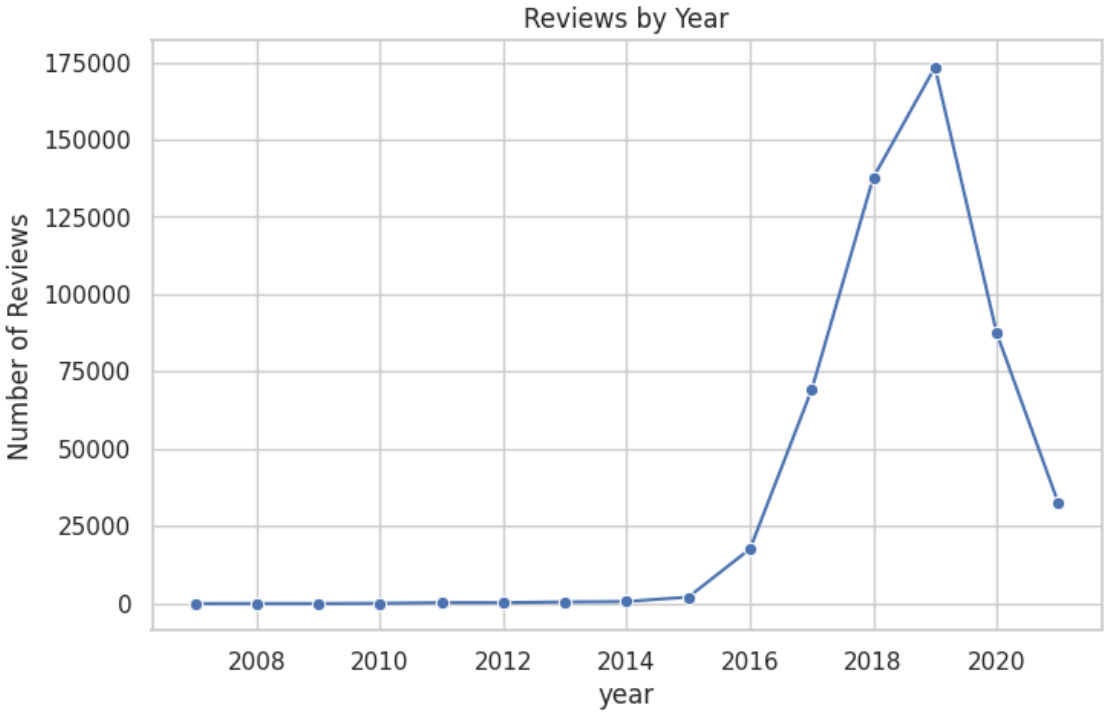
[102]:
```
# Reviews by month
month_order = ["January","February","March","April","May","June",
               "July","August","September","October","November","December"]

reviews_by_month = (
    merged_df.groupby("month")['user_id']
    .count()
    .reset_index(name = "review_count")
)

reviews_by_month['month'] = pd.Categorical(reviews_by_month['month'],
                                           categories = month_order,
                                           ordered = True)
reviews_by_month = reviews_by_month.sort_values("month")

plt.figure(figsize = (10,5))
sns.lineplot(data = reviews_by_month,
             x = "month",
             y = "review_count",
             marker = "o")
plt.title("Reviews by Month")
```

```
plt.ylabel("Number of Reviews")
plt.xticks(rotation = 45)
plt.show()
```



The most number of reviews are in the month of July, which is more than 55000.

[ ]:
```
# Reviews by year
reviews_by_year = (
    merged_df.groupby("year")['user_id']
    .count()
    .reset_index(name = "review_count")
)

plt.figure(figsize = (8,5))
sns.lineplot(data = reviews_by_year,
             x = "year",
             y = "review_count",
             marker = "o")
plt.title("Reviews by Year")
plt.ylabel("Number of Reviews")
plt.show()
```



Temporal patterns

By Weekday

We can see that Sunday has the most number of reviewers.

By Year

It is evident from the linechart, the year 2019 has had the most number of reviewers.

By Month

The most number of reviewers are in the month of July.

2.6 Question 1.6

2.6.1 *Question 1.6.1*

Write down your strategy of building the recommendation on business for customers in the markdown cell. You could create your own strategy or leverage the provided one here KNN on collaborative filtering.Please also include your strategy details in the report.

2.6.2 *Question 1.6.2*

Implementing the strategy code.

[ ]:
```
# 1.6.2 Collaborative Filtering with KNN ---
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix
import pandas as pd
import numpy as np

# Create a user-item matrix (pivot table)
user_item_matrix = merged_df.pivot_table(
    index='user_id',
    columns='business_name',
    values='rating',
    fill_value=0
)

# Convert to sparse matrix for efficient computation
sparse_user_item = csr_matrix(user_item_matrix.values)

# Build the KNN model on items (businesses)
# Using cosine similarity as it works well with sparse data
model_knn = NearestNeighbors(metric='cosine', algorithm='brute',␣
 ↪n_neighbors=20, n_jobs=-1)
model_knn.fit(sparse_user_item.T)  # Transpose to get item-item similarities

# Function to get business recommendations
def get_recommendations(business_name, n_recommendations=5):
    """
    Get similar businesses using KNN collaborative filtering
    """
    # Find the index of the requested business
    business_idx = user_item_matrix.columns.get_loc(business_name)

    # Find similar businesses
    distances, indices = model_knn.kneighbors(
        sparse_user_item.T[business_idx],
        n_neighbors=n_recommendations+1
    )

    # Get the list of similar businesses
    similar_businesses = []
    for i in range(1, len(indices.flatten())):
        similar_business = user_item_matrix.columns[indices.flatten()[i]]
        similarity = 1 - distances.flatten()[i]  # Convert distance to␣
 ↪similarity
        similar_businesses.append((similar_business, similarity))

    return similar_businesses
```

```python
# Test the recommendation system with a sample business
sample_business = user_item_matrix.columns[0]  # Get the first business as an␣
 ↪example
recommendations = get_recommendations(sample_business)

print(f"Businesses similar to '{sample_business}':")
for business, similarity in recommendations:
    print(f"- {business} (similarity: {similarity:.3f})")

# Generate recommendations for all businesses
business_recommendations = {}
for business in user_item_matrix.columns[:10]:  # Limit to first 10 for␣
 ↪demonstration
    business_recommendations[business] = get_recommendations(business)

print("\nSample of business recommendations:")
for business, recs in list(business_recommendations.items())[:3]:
    print(f"\nFor '{business}':")
    for rec, sim in recs:
        print(f"  - {rec} (similarity: {sim:.3f})")
```

```
Businesses similar to '1-800-GOT-JUNK? Anchorage':
- JBER Veterinary Clinic (similarity: 0.119)
- Mat-Su Urgent Care Palmer (similarity: 0.085)
- Aurora Military Housing - Richardson (similarity: 0.084)
- Alaska Scrap & Recycling (similarity: 0.080)
- Arctic Tails (similarity: 0.078)

Sample of business recommendations:

For '1-800-GOT-JUNK? Anchorage':
  - JBER Veterinary Clinic (similarity: 0.119)
  - Mat-Su Urgent Care Palmer (similarity: 0.085)
  - Aurora Military Housing - Richardson (similarity: 0.084)
  - Alaska Scrap & Recycling (similarity: 0.080)
  - Arctic Tails (similarity: 0.078)

For '108 Tap House & Burger Bar':
  - Local Grounds Coffee (similarity: 0.361)
  - Rain Country Nutrition (similarity: 0.319)
  - New York Cafe (similarity: 0.300)
  - Fat Stan's Sports Bar & Pizzeria (similarity: 0.287)
  - Bar Harbor Ale House (similarity: 0.266)

For '10th & M Seafoods':
  - Alaska Spine and Pain Center (similarity: 0.111)
  - Johnson's Tire Service- Midtown Anchorage (similarity: 0.108)
```

```
  - Rainbow Foods (similarity: 0.104)
  - Bentley's Porter House B & B (similarity: 0.102)
  - Mint Dental (similarity: 0.091)
```

Comments….

Please also include your analysis details in the report….

Give sufficient explanation in markdown about the results….

[ ]:
```python
# Evaluation of the KNN model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Create a train-test split for evaluation
train_data, test_data = train_test_split(merged_df, test_size=0.2,␣
 ↪random_state=42)

# Recreate the user-item matrix with training data only
train_matrix = train_data.pivot_table(
    index='user_id',
    columns='business_name',
    values='rating',
    fill_value=0
)

# Build KNN model on training data
train_sparse = csr_matrix(train_matrix.values)
model_knn_train = NearestNeighbors(metric='cosine', algorithm='brute',␣
 ↪n_neighbors=20)
model_knn_train.fit(train_sparse.T)

# Function to predict ratings (simplified)
def predict_rating(user_id, business_name):
    if business_name not in train_matrix.columns:
        return train_data['rating'].mean()  # Return global average for unknown␣
 ↪items

    business_idx = train_matrix.columns.get_loc(business_name)
    distances, indices = model_knn_train.kneighbors(train_sparse.
 ↪T[business_idx])

    # Find if the user has rated similar businesses
    user_ratings = train_matrix.loc[user_id]
    similar_businesses = train_matrix.columns[indices.flatten()]

    # Calculate weighted average of ratings for similar businesses
    weighted_sum = 0
    similarity_sum = 0
```

```
    for i, sim_business in enumerate(similar_businesses):
        if i == 0:  # Skip the business itself
            continue

        similarity = 1 - distances.flatten()[i]
        if user_ratings[sim_business] > 0:  # If user rated this similar␣
 ↪business
            weighted_sum += user_ratings[sim_business] * similarity
            similarity_sum += similarity

    if similarity_sum > 0:
        return weighted_sum / similarity_sum
    else:
        return train_data['rating'].mean()  # Return global average if no␣
 ↪similar ratings

# Make predictions on test set
test_data['predicted_rating'] = test_data.apply(
    lambda row: predict_rating(row['user_id'], row['business_name']),
    axis=1
)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test_data['rating'],␣
 ↪test_data['predicted_rating']))
print(f"Root Mean Squared Error: {rmse:.3f}")
```

Comments….

Please also include your analysis details in the report….

Give sufficient explanation in markdown about the results….

2.7 Question 1.7

2.7.1 *Question 1.7.1*

Building a visualization to explore the relationships of the rating and business categories.

[ ]:
```
### 1.7.1 Visualization of Ratings and Business Categories
import seaborn as sns
import matplotlib.pyplot as plt
import ast

# The 'category' column is a string representation of a list. We need to parse␣
 ↪it safely.
df_cat = merged_df.copy()
```

```
# Use a helper function to handle potential malformed strings in the category␣
 ↪column
def safe_literal_eval(val):
    try:
        return ast.literal_eval(val)
    except (ValueError, SyntaxError):
        return [] # Return empty list if parsing fails
df_cat['category_list'] = df_cat['category'].apply(safe_literal_eval)

# Explode the DataFrame to have one row per category for each business review
df_exploded = df_cat.explode('category_list')

# To make the plot readable, let's find the top 15 most frequent categories
top_15_categories = df_exploded['category_list'].value_counts().nlargest(15).
 ↪index

# Filter the DataFrame to only include these top categories
df_top_cat = df_exploded[df_exploded['category_list'].isin(top_15_categories)]

# Build the visualization: a box plot to show rating distributions
plt.figure(figsize=(12, 10))
sns.boxplot(data=df_top_cat, y='category_list', x='rating', orient='h')
plt.title('Distribution of Ratings Across Top 15 Business Categories')
plt.xlabel('Rating')
plt.ylabel('Business Category')
plt.tight_layout()
plt.show()
```

Comments….

Please also include your analysis details in the report….

Give sufficient explanation in markdown about the results….

2.7.2 *Question 1.7.2*

Finding out the actual reviews on lower ratings and analyzing on the reason.

[ ]:
```
### 1.7.2 Analysis of Lower Ratings
from wordcloud import WordCloud, STOPWORDS

# Filter for lower ratings (e.g., 1 and 2 stars)
low_rating_df = merged_df[merged_df['rating'] <= 2]

# Combine all review text from these low ratings into a single string
low_rating_text = ' '.join(low_rating_df['text'].dropna().astype(str))

# Define additional stopwords if necessary
```

```
custom_stopwords = set(STOPWORDS)
custom_stopwords.update(['good','review','place', 'food', 'one', 'get', 'go',␣
 ↪'now', 'will', 'restaurant', 'service'])

# Generate the word cloud
if low_rating_text and low_rating_text.strip():
    wordcloud = WordCloud(width=800, height=400, background_color='white',
                          stopwords=custom_stopwords, collocations=False).
 ↪generate(low_rating_text)

    plt.figure(figsize=(15, 7))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('Most Common Words in Low-Rated Reviews (1-2 Stars)')
    plt.show()
else:
    print("\nNo low-rated reviews with text to analyze.")
```

Comments....

Please also include your analysis details in the report....

Give sufficient explanation in markdown about the results....

2.8 Question 1.8

2.8.1 *Question 1.8.1*

Checking on the reviewer level reviewed business, sorting the review of each business by the review time (newtime column), and saving the business name into the list variable user_business_list for each reviewer.

[ ]:
```
### 1.8.1 Create User's Reviewed Business History List
# Group by reviewer (user_id), sort their reviews by time, and aggregate␣
 ↪business names into a list
# 1.8.1 - Create list of reviewed businesses per user, sorted by time

user_business_history = merged_df.groupby('user_id').apply(
    lambda x: x.sort_values('newtime')['business_name'].tolist()
).reset_index()
user_business_history.columns = ['user_id', 'user_business_list']

user_business_history.head(10)
```

Comments....

Please also include your analysis details in the report....

Give sufficient explanation in markdown about the results....

2.8.2 *Question 1.8.2*

Checking on the user_business_list, finding repeated business names for the same user

[ ]:
```
### 1.8.2 Remove Duplicated Business Names from History
# Store the original list length
# 1.8.2 - Remove duplicates and compare list lengths
def remove_duplicates_preserve_order(seq):
    seen = set()
    seen_add = seen.add
    return [x for x in seq if not (x in seen or seen_add(x))]

user_business_history['user_business_list_unique'] =␣
 ↪user_business_history['user_business_list'].
 ↪apply(remove_duplicates_preserve_order)

user_business_history['user_business_list_length_before'] =␣
 ↪user_business_history['user_business_list'].apply(len)
user_business_history['user_business_list_length_after'] =␣
 ↪user_business_history['user_business_list_unique'].apply(len)

user_business_history[['user_id', 'user_business_list_length_before',␣
 ↪'user_business_list_length_after']].head(10)
```

Comments....

Please also include your analysis details in the report....

Give sufficient explanation in markdown about the results....

2.8.3 *Question 1.8.3*

Checking on the user_business_list, finding the user similarities according to their past reviewed busines

[ ]:
```
### 1.8.2 Remove Duplicated Business Names from History
# Store the original list length
# 1.8.2 - Remove duplicates and compare list lengths
def remove_duplicates_preserve_order(seq):
    seen = set()
    seen_add = seen.add
    return [x for x in seq if not (x in seen or seen_add(x))]

user_business_history['user_business_list_unique'] =␣
 ↪user_business_history['user_business_list'].
 ↪apply(remove_duplicates_preserve_order)
user_business_history['list_length_before'] =␣
 ↪user_business_history['user_business_list'].apply(len)
```

```
user_business_history['list_length_after'] =␣
 ↪user_business_history['user_business_list_unique'].apply(len)

user_business_history[['user_id', 'list_length_before', 'list_length_after']].
 ↪head(10)
```

Comments....

Give sufficient explanation in markdown about the results....

3 *Part II*

3.1 *Question 2.1*

Exploring the review time series. Total reviews per day with review time (newtime from the dataframe) to form the review volume time series.

[ ]:
```
merged_df["newtime"] = pd.to_datetime(merged_df["newtime"], errors="coerce")

# Grouping reviews per day
daily_reviews = (
    merged_df.groupby(merged_df["newtime"].dt.date)
    .size()
    .reset_index(name="review_count")
)

# Convert back to datetime and set index
daily_reviews["newtime"] = pd.to_datetime(daily_reviews["newtime"])
daily_reviews.set_index("newtime", inplace=True)
```

Since there are some days not available in the review time series. We added those days to the review time series, using the default number of reviews and the mean value of the number of reviews per day across the entire dataset.

[ ]:
```
full_range = pd.date_range(
    start=daily_reviews.index.min(),
    end=daily_reviews.index.max(),
    freq="D"
)

# Fill missing days with mean number of reviews
# Compute mean reviews per day
mean_reviews = daily_reviews["review_count"].mean()

# Reindex and fill missing values
daily_reviews_filled = daily_reviews.reindex(full_range, fill_value=np.nan)
daily_reviews_filled["review_count"] = daily_reviews_filled["review_count"].
 ↪fillna(mean_reviews)
```

```
daily_reviews_filled
```

Decomposing the submission review time series with addictive mode and analyses on the results to find if there is any seasonality pattern.

[ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

ts_df = daily_reviews_filled[['review_count']]

# Additive Decomposition
add_result = seasonal_decompose(ts_df['review_count'], model='additive',␣
 ↪period=7)  # 7 days = weekly seasonality
plt.rcParams["figure.figsize"] = (12, 8)
add_result.plot().suptitle('Review Count - Additive Decomposition', fontsize=12)
plt.show()
```

We apply Additive Decomposition, assuming the seasonal effect is constant across time, since the reviews are provided from multiple business categories.

Analysis of Review Time Series Decomposition

Observed top Panel:

The observed series shows the raw daily review counts.

Reviews were very low before 2012, but after that, there was a steady increase.

The peak occurred around 2018–2019, followed by a decline around 2020 (possibly linked to the COVID-19 pandemic reducing business activity).

Trend:

The trend confirms that the overall number of reviews has grown significantly over time, especially from 2014 onwards.

The highest trend values are between 2017 and 2019, where review activity was at its peak.

After 2019, the trend shows a clear decline.

Seasonal Component:

The seasonal pattern appears between roughly -5 and +5. This suggests that reviews do not have strong daily/weekly seasonality at the aggregated dataset level.

However, it's possible that seasonality exists at finer levels per business category or region, but it is averaged out in the overall dataset.

Residuals:

The residuals show a lot of random variation around the trend.

3.2 *Question 2.2*

Using time series model ARIMA for forecasting the future. Finding the best model with different parameters on ARIMA model.

[ ]:
```
from itertools import product
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error

# Train/Test split (last 20% for testing)
split_idx = int(len(daily_reviews_filled) * 0.8)
train, test = daily_reviews_filled.iloc[:split_idx], daily_reviews_filled.
 ↪iloc[split_idx:]
```

Finding out the best model with lowest Mean Absolute Error from 27 choices to train and test with yourself with grid search. The parameter range for p,d,q are all from [0, 1, 2].

[ ]:
```
# Grid search over p,d,q
p = d = q = [0, 1, 2]
best_order, best_mae, best_model = None, float('inf'), None

for order in product(p, d, q):
    try:
        model = ARIMA(train, order=order)
        fit = model.fit()
        forecast = fit.forecast(steps=len(test))
        mae = mean_absolute_error(test, forecast)

        if mae < best_mae:
            best_mae = mae
            best_order = order
            best_model = fit
    except Exception as e:
        # Some parameter combos may not converge; skip them
        continue

print(f"\nBest order (p,d,q): {best_order}  |  MAE: {best_mae:.3f}")
```

The model with the lowest MAE was selected as the best-performing configuration and the optimal order was found to be (2, 0, 2), achieving an MAE of 127.765, indicating relatively strong predictive accuracy for this dataset.

[ ]:
```
# Plot forecast
forecast = best_model.forecast(steps=len(test))

plt.figure(figsize=(14,6))
plt.plot(train, label="Train")
plt.plot(test, label="Test", color='orange')
```

```
plt.plot(forecast, label="Forecast", color='green')
plt.title(f"ARIMA{best_order} - Forecast vs Actual")
plt.xlabel("Date")
plt.ylabel("Daily Reviews")
plt.legend()
plt.show()
```

Exploring the deep learning time series forecasting methods such as LSTM and RNN. Please write down your discussion around the necessary data wrangling and modeling steps (steps on how to achieve, not actual code).

Also please give the reference of the deep learning time series forecasting models you are using. Please also include your discussion details and implementation in the report.

Exploring Deep Learning Time-Series Forecasting (LSTM & RNN)

Traditional models like ARIMA are effective for linear, stationary time-series data. However, they face limitations when dealing with:

Long-term dependencies

Non-linear or irregular seasonality

Multiple exogenous variables

To address these challenges, deep learning models, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks offer a powerful alternative. These architectures can learn complex temporal dynamics and capture long-range relationships in review volume data.

Modeling steps

- Effective time-series forecasting with neural networks requires meticulous preprocessing:
- Data Cleaning & Completion
- Resampling & Aggregation
- Standardize to a consistent frequency
- Normalization / Scaling
- Apply Min–Max or Standard scaling to stabilize training gradients
- Dataset Splitting
- Divide into train, validation, and test sets
- Reshape if require for Model Input

Modeling Strategy

- Model Selection
  - Begin with a basic RNN
  - Progress to LSTM for improved long-term memory retention
- Architecture Example

  - Input layer → LSTM layers (with dropout) → Dense output layer
- Loss Functions & Optimizers
  - Use MAE or MSE for loss
  - Adam or RMSprop for optimization
- Training
  - Implement early stopping to prevent overfitting
  - Monitor validation loss throughout
- Evaluation & Forecasting
  - Compare performance metrics (MAE, RMSE) against ARIMA
  - Generate and visualize forecasts on the hold-out set

3.3 *Question 2.3*

Analizing report file by Universities Australia via Indigenous Strategy annual report.

Data Extraction Carefully review the PDF and identify all relevant quantitative data, tables, and figures that can be extracted or digitized; Present any extracted data in a structured format (e.g., CSV, Excel table, or DataFrame);

[ ]:
```
!pip install pdfplumber camelot-py[cv] tabula-py

# Install required libraries
# pdfplumber is an easy way to copy text/tables.
# camelot-py / tabula-py needed for extracting tables from PDF.
```

[ ]:
```
import os
import io
import requests
import camelot

# Download the PDF
pdf_url = "https://universitiesaustralia.edu.au/wp-content/uploads/2022/08/
 ↪UA_Indigenous_Strategy_Annual_Report_May-2022.pdf"
local_pdf_path = "/content/UA_Indigenous_Strategy_Annual_Report_May-2022.pdf"

response = requests.get(pdf_url)
pdf_bytes = io.BytesIO(response.content)
```

After reviewing the PDF, we found that the most relevant information was concentrated in the tables and figures. For that reason, it was important to identify how many tables and figures were present throughout the document, so we could later extract the data and determine which ones were truly meaningful.

[ ]:
```
import re
import pdfplumber

def find_all_labels(pdf_bytes, keyword="Table", suffix=":"):
    """
    Extract mentions of a keyword (e.g., 'Table' or 'Figure') followed by up to␣
 ↪4 characters.
    Returns a list of unique labels that end with the specified suffix.
    """
    labels = set()
    pattern = rf"({keyword}.{{0,4}})"

    with pdfplumber.open(pdf_bytes) as pdf:
        for page in pdf.pages:
            text = page.extract_text() or ""
            matches = re.findall(pattern, text)
            for m in matches:
                label = m.strip()
                if label.endswith(suffix):
                    labels.add(label)

    # Sort labels numerically by the first number found in each label
    def extract_number(label):
        match = re.search(r"\d+", label)
        return int(match.group()) if match else float('inf')

    return sorted(labels, key=extract_number)
```

[ ]:
```
# Finding tables in the pdf
table_labels = find_all_labels(pdf_bytes, keyword="Table")

print("Tables found in PDF:")
table_labels
```

[ ]:
```
# Find figures in the pdf
figure_labels = find_all_labels(pdf_bytes, keyword="Figure")

print("Figures found in PDF:\n")
figure_labels
```

[ ]:
```
# Download if not already prevents downloading it again every time we re-run␣
 ↪the cell.
if not os.path.exists(local_pdf_path):
    r = requests.get(pdf_url)
    with open(local_pdf_path, "wb") as f:
        f.write(r.content)
```

```
# We used Camelot to Extract tables
tables = camelot.read_pdf(local_pdf_path, pages='all', flavor='stream')

print(f"Found {tables.n} tables")

# Then we Save each table as CSV
tables_info = []
for i, tbl in enumerate(tables):
    table_df = tbl.df
    csv_path = f"/content/table_camelot_{i}.csv"
    table_df.to_csv(csv_path, index=False)
    tables_info.append({"table_index": i, "rows": table_df.shape[0], "cols":␣
 ↪table_df.shape[1], "csv": csv_path})
```

After applying table extraction, we were able to recover meaningful information from the main table within the document (Table 1: Indigenous enrolments, by course level, 2008 and 2020). However, several additional tables were also generated, many of which contained unclear or irrelevant data. To ensure accuracy and usability, we proceeded to clean and refine the data extracted from the primary table.

[ ]:
```
for idx in [13]:
    path = f"/content/table_camelot_{idx}.csv"
    print(f"\n--- Table {idx} ---")
    display(pd.read_csv(path).head(20))
```

Cleaning code from Table 1

[ ]:
```
idx = 13
path = f"/content/table_camelot_{idx}.csv"
raw = pd.read_csv(path, header=None)

indigenous_rep_df = raw.iloc[5:12].copy()       # Only kepp rows 4 to 12
indigenous_rep_df.columns = ["Course level", "2008", "2020", "Growth since␣
 ↪2008", "Annual avg growth since 2008"]
indigenous_rep_df.reset_index(drop=True, inplace=True)


for col in ["2008", "2020"]:
    indigenous_rep_df[col] = indigenous_rep_df[col].astype(str).str.
 ↪replace(",", "", regex=False)
    indigenous_rep_df[col] = pd.to_numeric(indigenous_rep_df[col],␣
 ↪errors="coerce")

for col in [ "Annual avg growth since 2008"]:
    indigenous_rep_df[col] = indigenous_rep_df[col].astype(str).str.
 ↪replace(r"\n", "", regex=True)
```

```
print("Clean Table 1: Indigenous enrolments, by course level, 2008 and 2020\n")
indigenous_rep_df
```

Since we have more informtion related to the figures was necessary implement other data extarction tecniques

[ ]:
```
# Importing libraries to extract data from PDF

!pip install pdfplumber PyMuPDF openpyxl

import pdfplumber
import fitz  # PyMuPDF
```

Defining functions to extract information from figures

After reviewing each figure in detail, only the most relevant ones were selected for analysis. Many of the excluded figures featured bar charts without specific percentage values assigned to each year, making it difficult to extract meaningful data. Additionally, figures that presented information for a single year lacked the temporal context necessary for comparison across time, limiting their analytical value.

[ ]:
```
# Defining the list of figures with most relevant information
target_figures = [
    "Figure 2:",
    "Figure 10:",
    "Figure 11:",
    "Figure 12:",
    "Figure 24:"
]

# Filter figure_labels to keep only the ones in target_figures
filtered_figures = [f for f in figure_labels if f in target_figures]
```

[ ]:
```
# Now we need to identify which pages include figures.
def find_pages_with_caption(pdf_bytes, caption_pattern):
    """Find pages containing a given caption text (e.g., 'Figure 1')."""
    pages_with_caption = []
    with pdfplumber.open(pdf_bytes) as pdf:
        for i, page in enumerate(pdf.pages):
            text = page.extract_text() or ""
            if re.search(caption_pattern, text, flags=re.I):
                pages_with_caption.append((i, text))
    return pages_with_caption
```

[ ]:
```
# Dictionary to store results: {label: [(page_number, page_text), ...]}
figures_with_pages = {}

for label in filtered_figures:
```

```
    matches = find_pages_with_caption(pdf_bytes, caption_pattern=label)
    if matches:  # Only store if there's at least one match
        figures_with_pages[label] = matches
```

[ ]:
```
for label, pages in figures_with_pages.items():
    print(f"{label} found on page(s): {[p[0] for p in pages]}")
```

[ ]:
```
# Since most of the figures are represented as percentages, we need a function␣
 ↪to extract those values accurately.
def extract_percentages_from_text(text):
    """Extract all % numbers (positive/negative floats)."""
    nums = re.findall(r'(-?\d+(?:\.\d+)?)\s*%', text)
    return [float(x) for x in nums]

'''def extract_values_from_text(text):
    """Extract all % numbers (positive/negative floats)."""
    nums = re.findall(r'(-?\d+(?:\.\d+)?)\s*%', text)
    return [float(x) for x in nums]'''
```

[ ]:
```
def try_map_to_years(years, numbers):
    """Map extracted numbers to one or two series."""
    n_years = len(years)
    if len(numbers) == n_years:
        return {'series1': numbers}
    if len(numbers) == 2 * n_years:
        return {'series1': numbers[:n_years], 'series2': numbers[n_years:]}
    if len(numbers) >= n_years:
        return {'series1': numbers[:n_years], 'series2': numbers[n_years:␣
 ↪n_years*2]}
    return {}
```

[ ]:
```
def find_all_labels(pdf_bytes, keyword="Figure", suffix=","):
    """
    Extract mentions of a keyword (e.g., 'Table' or 'Figure') followed by up to␣
 ↪4 characters.
    Returns a list of unique labels that end with the specified suffix.
    """
    labels = set()
    pattern = rf"({keyword}.{{0,4}})"

    with pdfplumber.open(pdf_bytes) as pdf:
        for page in pdf.pages:
            text = page.extract_text() or ""
            matches = re.findall(pattern, text)
            for m in matches:
                label = m.strip()
                if label.endswith(suffix):
```

```
                    labels.add(label)

    # Sort labels numerically by the first number found in each label
    def extract_number(label):
        match = re.search(r"\d+", label)
        return int(match.group()) if match else float('inf')

    return sorted(labels, key=extract_number)

# Find figures in the pdf
figure_titles = find_all_labels(pdf_bytes, keyword="Figure")

print("Figures found in PDF:\n")
figure_titles
```

[ ]:
```
target_figures = [
    "Figure 2:",
    "Figure 3:",
    "Figure 11:",
    "Figure 12:"
]

all_figures_data = {}  # dictionary to keep all figures

# Loop through each target figure
for target in target_figures:
    print(f"\n Looking for {target}")

    found = find_pages_with_caption(pdf_bytes, target)

    for page_no, text in found:
        percentages = extract_percentages_from_text(text)

        # Year ranges
        if target in ["Figure 2:"]:
            years = list(range(2006, 2021))
        elif target == "Figure 3:":
            years = list(range(2007, 2021))
        elif target == "Figure 5:":
            years = list(range(2013, 2022))
        elif target == "Figure 11:":
            years = list(range(2005, 2013))
        elif target == "Figure 12:":
            years = list(range(2005, 2018))
        else:
            years = list(range(2010, 2020))  # fallback
```

```
        mapped = try_map_to_years(years, percentages)

        try:
            if 'series1' in mapped:
                df = pd.DataFrame({
                    'Year': years,
                    'Series_1': mapped['series1']
                })
            else:
                df = pd.DataFrame({'Value': percentages})

            print(df)

            # Save this figure's DataFrame in dictionary
            all_figures_data[target] = df

        except ValueError as e:
            print(f"  Could not create DataFrame for {target} due to mismatch:␣
↪{e}")
```

Clean Figure 2 after extraction

[ ]:
```
# Get Figure 2 dataframe (assuming it's in your dict all_figures_data)
figure2_df = all_figures_data["Figure 2:"].copy()

# Clean column name + sort ascending
figure2_df = figure2_df.rename(
    columns={"Series_1": "Indigenous growth Enrolment (%)"}
)

# Sorting values
figure2_df = figure2_df.sort_values(
    by="Indigenous growth Enrolment (%)"
).reset_index(drop=True)

print(" Cleaned Figure 2:")
figure2_df
```

Clean Figure 3 after extraction

[ ]:
```
# Get Figure 3 dataframe
figure3_df = all_figures_data["Figure 3:"].copy()

# Rename column
figure3_df = figure3_df.rename(
    columns={"Series_1": "Undergraduate Enrolment (%)"}
)
```

```
# Correct order you want
correct_order_f3 = [5.8, 1.2, 7.6, 9.2, 7.7, 6.8, 9.3, 9.4,
                    8.7, 8.9, 8.6, 3.2, 4.2, 7.4]

# Replace with correct order
figure3_df["Undergraduate Enrolment (%)"] = correct_order_f3

# Show result
print(" Cleaned Figure 3:")
figure3_df
```

Clean Figure 11 after extraction

[ ]:
```
# Get Figure 12 dataframe
figure11_df = all_figures_data["Figure 11:"].copy()

# Renaming column
figure11_df = figure11_df.rename(
    columns={"Series_1": "Nine-year Enrolment (%)"}
)

# Correct order you want
correct_order_f11 = [46.5, 47.2, 47.1, 47.9, 47.7, 47.8, 47.5, 49.4]

# Replace with correct order
figure11_df["Nine-year Enrolment (%)"] = correct_order_f11

# Show result
print(" Cleaned Figure 11:")
figure11_df
```

Clean Figure 12 after extraction

[ ]:
```
# Get Figure 12 dataframe
figure12_df = all_figures_data["Figure 12:"].copy()

# Rename column
figure12_df = figure12_df.rename(
    columns={"Series_1": "Indig Bachelor Enrolment (%)"}
)

# Correct order you want
correct_order_f12 = [25.8, 21.9, 22.9, 20.0, 19.9, 20.7,
                20.7, 20.2, 19.1, 19.5, 18.5, 18.4, 20.4]

# Replace with correct order
figure12_df["Indig Bachelor Enrolment (%)"] = correct_order_f12
```

```
# Show result
print(" Cleaned Figure 12:")
figure12_df
```

Data Analysis Discovering common patterns or trends from the Indigenous Strategy annual report

Combining figures data from the indigenous report into one table.

[ ]:
```
final_df = (
    figure2_df
    .merge(figure3_df, on="Year", how="outer")
    .merge(figure11_df, on="Year", how="outer")
    .merge(figure12_df, on="Year", how="outer")
)

# Replacing missing values with the mean of each column
final_df = final_df.fillna(final_df.mean(numeric_only=True))
final_df = final_df.round(2)

print("Final DataFrame with NaNs replaced by column means:")
final_df
```

[ ]:
```
final_df.set_index("Year").plot(
    kind="line", marker="o", figsize=(15, 6)
)
plt.title("Enrolment Trends by Year")
plt.xlabel("Year")
plt.ylabel("Percentage (%)")
plt.legend(title="Series",bbox_to_anchor=(1.25, 1), loc="upper right",␣
 ↪frameon=False)
plt.grid(True, alpha=1)
plt.tight_layout()
plt.show()
```

Insights Providing a clear and concise summary of the main patterns, trends, or correlations discovered from the analysis

Main Patterns

Indigenous Enrolment (%):

Rose slightly from 1.52% in 2005 to around 1.25% in 2020.

After peaking in 2006 (2.04%), it declined gradually and stabilised near 1.2%.

Undergraduate Enrolment (%):

Showed greater volatility: a sharp dip in 2008 (1.2%), a strong recovery (9–9.4% in 2013–14), then a drop in 2018 (3.2%) before recovering to 7.4% in 2020.

Nine-year Completion Enrolment (%):

Remained the most stable indicator, holding between 46.5% and 49.4% across the whole period.

Suggests steady retention/completion rates despite enrolment fluctuations.

Indigenous Bachelor Enrolment (%):

Declined from 25.8% (2005) to a low of 18.4% (2016), then increased to about 20.6% (2017–20).

There has been steady growth in higher-level qualifications (Bachelor and Postgraduate), yet Indigenous participation at entry-level remains around 1–1.3% of total enrolments.

While completion rates have remained stable, attention may need to shift toward boosting initial enrolments and strengthening access pathways such as Enabling and Sub-bachelor programs.

The sharp rise in non-award and postgraduate enrolments highlights strong demand for lifelong learning. However, ensuring these pathways effectively lead into formal degree programs remains a critical priority.

4 *References*

Australia, U., 2020. Indigenous strategy annual report. Retrieved Sep, 11, p.2025.

Siami-Namini, S., Tavakoli, N. and Namin, A.S., 2018, December. A comparison of ARIMA and LSTM in forecasting time series. In 2018 17th IEEE international conference on machine learning and applications (ICMLA) (pp. 1394-1401). Ieee.