USER ENGAGEMENT ANALYSIS FOR RESTAURANT SUCCESS

INTRODUCTION

Founded in 2004, Yelp is a widely used online directory that helps users find local businesses, including bars, restaurants, cafes, hairdressers, spas, and gas stations.

Yelp can be accessed via its website or official iOS and Android apps. Users can search for businesses by type and filter results by location, price range, and specific features like outdoor seating, delivery services, or reservations.

Yelp's platform has a strong social component, encouraging users to leave written reviews, star ratings, and photos of their experiences. Users can connect with friends through Facebook and their device's address book. Reviews can be rated by others, and top reviewers may earn Yelp Elite status (Stephenson, 2020).

PROBLEM STATEMENT

In the highly competitive restaurant industry, it's essential for stakeholders to grasp the elements that drive business success. This project leverages the Yelp dataset to explore how user engagement measured through reviews, tips, and check-ins correlates with key success metrics like review counts and ratings for restaurants. By analyzing these relationships, the project seeks to provide insights that can help restaurant owners and managers enhance their strategies for attracting and retaining customers.

Research Objectives

- **Correlate user engagement with reviews and ratings**: Determine if higher engagement (reviews, tips, check-ins) correlates with increased review counts and higher average ratings.
- **Impact of sentiment on reviews and ratings**: Investigate whether positive review and tip sentiment leads to higher star ratings and influences the number of reviews.
- **Time trends in engagement**: Explore if consistent user engagement over time is a better indicator of long-term success compared to sporadic activity bursts.

Hypothesis

- Increased user engagement, indicated by a higher number of reviews, tips, and check-ins, correlates with elevated review counts and ratings for restaurants.
- Positive sentiment in reviews and tips enhances overall ratings and review counts for restaurants.
- Steady engagement over time is positively linked to ongoing business success for restaurants.

Dataset Overview

- This dataset, a subset of Yelp's data, contains information about businesses in eight metropolitan areas across the USA and Canada.

- The original data, provided by Yelp, is available in five JSON files: business, review, user, tip, and check in, stored in a database for convenient data retrieval.
- The dataset is directly available for download from either Yelp Dataset or Kaggle.

DATABASE CREATION

Importing Libraries

```
import pandas as pd
import json
from sqlalchemy import create_engine #For creating database engine
```

The following code imports the necessary libraries, setting up the environment to seamlessly read and process data from JSON files into Pandas DataFrame, facilitating efficient analysis and manipulation tasks thereafter.

- ***pandas*** for data manipulation and analysis.
- ***Json*** for parsing JSON data.
- ***sqlalchemy.create_engine*** for creating a database engine

Loading JSON files into DataFrame

```
with open('yelp_academic_dataset_business.json', 'r', encoding='utf-8') as f:
    business_data = [json.loads(line) for line in f]
business_df = pd.DataFrame(business_data)

with open('yelp_academic_dataset_checkin.json', 'r', encoding='utf-8') as f:
    checkin_data = [json.loads(line) for line in f]
checkin_df = pd.DataFrame(checkin_data)

with open('yelp_academic_dataset_review.json', 'r', encoding='utf-8') as f:
    review_data = [json.loads(line) for line in f]
review_df = pd.DataFrame(review_data)

with open('yelp_academic_dataset_tip.json', 'r', encoding='utf-8') as f:
    tip_data = [json.loads(line) for line in f]
tip_df = pd.DataFrame(tip_data)

with open('yelp_academic_dataset_user.json', 'r', encoding='utf-8') as f:
    user_data = [json.loads(line) for line in f]
user_df = pd.DataFrame(user_data)
```

```
print(business_df.shape)
print(checkin_df.shape)
print(review_df.shape)
print(tip_df.shape)
print(user_df.shape)
```

```
(150346, 14)
(131930, 2)
(6990280, 9)
(908915, 5)
(1987897, 22)
```

The above code snippet demonstrates loading data from five JSON files obtained from Yelp into Pandas DataFrames, facilitating efficient data manipulation and analysis. Each JSON file is read line-by-line and converted into a list of dictionaries using ***json.loads()***. These dictionaries are then used to create Pandas DataFrames ***(business_df, checkin_df, review_df, tip_df,*** and ***user_df***).

After loading each DataFrame, the ***.shape*** attribute is used to print the number of rows and columns for each dataset, providing an immediate overview of its size and structure:

- ***business_df***: Contains 150,346 rows and 14 columns.
- ***checkin_df***: Contains 131,930 rows and 2 columns.
- ***review_df***: Contains 6,990,280 rows and 9 columns.
- ***tip_df***: Contains 908,915 rows and 5 columns.
- ***user_df***: Contains 1,987,897 rows and 22 columns.

```
business_df.drop(['attributes','hours'],axis = 1, inplace = True)
```

The following code snippet modifies the ***business_df*** DataFrame by removing the columns ***attributes*** and ***hours*** directly (specified with ***axis=1***, denoting columns), using Pandas' ***.drop()*** method. Setting ***inplace=True*** ensures that the DataFrame is altered in place, avoiding the need to create a new DataFrame instance.

These columns have been removed as they are deemed irrelevant because they contain details unrelated to the project's focus on analysing user engagement for business success, ensuring that the remaining data aligns more closely with the project objectives.

Establishing Database Connection

```
engine = create_engine('sqlite:///yelp.db')

def load_dataframe(df, table_name, engine):
    df.to_sql(table_name, con=engine, if_exists='replace', index=False)

# Load each DataFrame into a separate table
load_dataframe(business_df, 'business', engine)
load_dataframe(review_df, 'review', engine)
load_dataframe(user_df, 'user', engine)
load_dataframe(tip_df, 'tip', engine)
load_dataframe(checkin_df, 'checkin', engine)
```

The above code snippet connects to a SQLite database named ***yelp.db*** using SQLAlchemy's ***create_engine*** function, storing this connection in the variable ***engine***. It defines ***load_dataframe(df, table_name, engine)*** to efficiently transfer the DataFrames into SQLite tables via the engine. The function uses ***df.to_sql(table_name, con=engine, if_exists='replace', index=False)*** where ***replace*** overwrites existing tables and ***index=False*** omits the DataFrame's index from the database table.

Each of the five DataFrames is then loaded into SQLite as separate tables (***'business'***, ***'review'***, ***'user'***, ***'tip'***, ***'checkin'***) using ***load_dataframe()***. This approach effectively organizes the Yelp dataset into a structured SQLite database format, facilitating easy querying, manipulation, and analysis via SQL or Python.

ANALYSIS AND FINDINGS

Importing Libraries

```python
import pandas as pd  # Data manipulation and analysis
import matplotlib.pyplot as plt  # Creating static, animated, and interactive visualizations
import seaborn as sns  # Statistical graphics: attractive and informative
from datetime import datetime  # Manipulate dates and times
import numpy as np  # Numerical computations and handling arrays
import sqlite3  # Connect to and interact with SQLite databases
import folium  # Building interactive maps
from geopy.geocoders import Nominatim  # Used for mapping locations and analyzing spatial data
from matplotlib.colors import LinearSegmentedColormap  # Creating custom colormaps
from IPython.display import display  # Displaying rich media
import warnings
warnings.filterwarnings('ignore')
```

The following libraries were imported for the purpose of analysis. The code snippet includes comments explaining each library and its specific use.

Database Connection

```python
# Creating database connection
conn = sqlite3.connect('yelp.db')
```

```python
# Tables in the database
tables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE type='table'",conn)
tables
```

	name
0	business
1	review
2	user
3	tip
4	checkin

The above code snippet establishes a connection to SQLite database named ***yelp.db*** using ***sqlite3.connect***. It then retrieves the names of all tables in the database by executing an SQL query (as shown), storing the result accordingly and displaying the output which lists the tables: ***'business'***, ***'review'***, ***'user'***, ***'tip'***, and ***'checkin'***.

Displaying Table Data

```python
# Displaying the data from each table
for table in tables['name']:
    print('-'*51,f'{table}','-'*51)
    display(pd.read_sql_query(f"select * from {table} limit 5",conn))
    print("\n")
```

This code iterates over each table name in the ***tables*** DataFrame, printing a formatted header with the table name and displaying the first 5 rows of each table. It achieves this by executing an SQL query to retrieve the data and loading the results into a Pandas DataFrame for display. This provides a quick preview of the data in each SQLite database table.

Below is a brief explanation of the table output:

1. Business table

--- business ---

	business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open	categories
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-119.711197	5.0	7	0	Doctors, Traditional Chinese Medicine, Naturop...
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	-90.335695	3.0	15	1	Shipping Centers, Local Services, Notaries, Ma...
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-110.880452	3.5	22	0	Department Stores, Shopping, Fashion, Home & G...
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.955505	-75.155564	4.0	80	1	Restaurants, Food, Bubble Tea, Coffee & Tea, B...
4	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA	18054	40.338183	-75.471659	4.5	13	1	Brewpubs, Breweries, Food

The table contains 12 columns: Business ID, name, address, city, state, postal code, latitude, longitude, stars, review count, is_open, and categories. The Business ID serves as the primary key, uniquely identifying each business. However, multiple businesses can share the same name, indicating different branches. The categories column lists various categories, but only businesses categorized as restaurants will be considered. The table provides comprehensive details about each business, including its location (address, city, state, postal code, latitude, longitude), average rating (stars), total number of reviews (review count), and operational status (is_open).

2. Review table

-- review --

	review_id	user_id	business_id	stars	useful	funny	cool	text	date
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	0	0	0	If you decide to eat here, just be aware it is...	2018-07-07 22:09:11
1	BiTunyQ73aT9WBnpR9DZGw	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	5.0	1	0	1	I've taken a lot of spin classes over the year...	2012-01-03 15:28:18
2	saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3.0	0	0	0	Family diner. Had the buffet. Eclectic assortm...	2014-02-05 20:30:30
3	AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5.0	1	0	1	Wow! Yummy, different, delicious. Our favo...	2015-01-04 00:01:03
4	Sx8TMOWLNuJBWer-0pcmoA	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	4.0	1	0	1	Cute interior and owner (?) gave us tour of up...	2017-01-14 20:54:15

The table contains the following columns: review_id, user_id, business_id, stars, useful, funny, cool, text, and date. The review_id serves as the primary key, while user_id and business_id are foreign keys. This table represents a one-to-many relationship, detailing reviews associated with various users and businesses.

3. User table

-- user --

	user_id	name	review_count	yelping_since	useful	funny	cool	elite	friends	fans	...	comp
0	qVc8ODYU5SZjKXVBgXdI7w	Walker	585	2007-01-25 16:47:26	7217	1259	5994	2007	NSCy54eWehBJyZdG2iE84w, pe42u7DcCH2QmI81NX-8qA...	267	...	
1	j14WgRoU_-2ZE1aw1dXrJg	Daniel	4333	2009-01-25 04:35:42	43091	13066	27281	2009,2010,2011,2012,2013,2014,2015,2016,2017,2...	ueRPE0CX75ePGMqOFVj6IQ, 52oH4DrRvzzl8wh5UXyU0A...	3138	...	
2	2WnXYQFK0hXEoTxPtV2zvg	Steph	665	2008-07-25 10:41:00	2086	1010	1003	2009,2010,2011,2012,2013	LuO3Bn4f3rlhyHIaNfTlnA, j9B4XdHUhDfTKVecyWQgyA...	52	...	
3	SZDeASXq7o05mMNLshsdIA	Gwen	224	2005-11-29 04:38:33	512	330	299	2009,2010,2011	enx1vVPnfdNUdPho6PH_wg, 4wOcvMLtU6a9Lslggq74Vg...	28	...	
4	hA5lMy-EnncsH4JoR-hFGQ	Karen	79	2007-01-05 19:40:59	29	15	7		PBK4q9KEEBHhFvSXCUirlw, 3FWPpM7KU1gXeOM_ZbYMbA...	1	...	

5 rows × 22 columns

The table contains the following columns: user_id, name, review_count, yelping_since, useful, funny, cool, elite, friends, fans, compliment_more, compliment_profile, compliment_cute, compliment_list, compliment_note, compliment_plain, compliment_cool, compliment_funny, compliment_writer, and compliment_photos. The user_id serves as the primary key. This table also includes information about elite users, identified by their activity and badges.

4. Tip table

-- tip --

	user_id	business_id	text	date	compliment_count
0	AGNUgVwnZUey3gcPCJ76iw	3uLgwr0qeCNMjKenHJwPGQ	Avengers time with the ladies.	2012-05-18 02:17:21	0
1	NBN4MgHP9D3cw--SnauTkA	QoezRbYQncpRqyrLH6lqjg	They have lots of good deserts and tasty cuban...	2013-02-05 18:35:10	0
2	-copOvldyKh1qr-vzkDEvw	MYoRNLb5chwjQe3c_k37Gg	It's open even when you think it isn't	2013-08-18 00:56:08	0
3	FjMQVZjSqY8syIO-53KFKw	hV-bABTK-glh5wj31ps_Jw	Very decent fried chicken	2017-06-27 23:05:38	0
4	Id0AperBXk1h6UbqmM80zw	_uN0OudeJ3ZI_tf6nxg5ww	Appetizers.. platter special for lunch	2012-10-06 19:43:09	0

The table contains the following columns: user_id, business_id, text, date, and compliment_count. Both user_id and business_id serve as foreign keys. This table captures tips, where multiple users can give tips to multiple businesses.

5. Checkin table

-- checkin --

	business_id	date
0	---kPU91CF4Lq2-WlRu9Lw	2020-03-13 21:10:56, 2020-06-02 22:18:06, 2020...
1	--0iUa4sNDFiZFrAdIWhZQ	2010-09-13 21:43:09, 2011-05-04 23:08:15, 2011...
2	--30_8IhuyMHbSOcNWd6DQ	2013-06-14 23:29:17, 2014-08-13 23:20:22
3	--7PUidqRWpRSpXebiyxTg	2011-02-15 17:12:00, 2011-07-28 02:46:10, 2012...
4	--7jw19RH9JKXgFohspgQw	2014-04-21 20:42:11, 2014-04-28 21:04:46, 2014...

The table contains the following columns: business_id and date (in raw string format). The data needs to be separated and analysed for further insights.

Data Analysis and Visualization

Analysis of Open Restaurant Businesses Among Total Businesses

```python
# Total Business Count
pd.read_sql_query("select count(*) from business", conn)
```

	count(*)
0	150346

```python
# Open Restaurant Business
restaurant_business = pd.read_sql_query("select business_id, review_count from business WHERE LOWER(categories) LIKE '%restaurant%' and is_open = 1",conn)
len(restaurant_business)
```

```
35004
```

Out of 150K businesses, 35K are open restaurant business.

The provided code snippets aim to analyse the distribution and success metrics of businesses listed in the database.

Out of the total 150,346 businesses, **35,004** are categorized as open restaurant businesses. The results include a table showing the distribution of business success metrics, specifically focusing on the review_count for these open restaurant businesses.

```python
# Descri;ptive Statistics for review_count and star columns
pd.read_sql_query(f"""SELECT
  AVG(review_count) AS average_review_count,
  MIN(review_count) AS min_review_count,
  MAX(review_count) AS max_review_count,
  (SELECT review_count FROM business ORDER BY review_count LIMIT 1 OFFSET (SELECT COUNT(*) FROM business) / 2) AS median_review_count,

  AVG(stars) AS average_star_rating,
  MIN(stars) AS min_star_rating,
  MAX(stars) AS max_star_rating,
  (SELECT stars FROM business ORDER BY stars LIMIT 1 OFFSET (SELECT COUNT(*) FROM business) / 2) AS median_star_rating

FROM business
WHERE business_id IN {tuple(restaurant_business['business_id'])};

""",conn).transpose()
```

	0
average_review_count	104.097789
min_review_count	5.000000
max_review_count	7568.000000
median_review_count	15.000000
average_star_rating	3.523969
min_star_rating	1.000000
max_star_rating	5.000000
median_star_rating	3.500000

The above code calculates descriptive statistics (average, minimum, maximum, and median) for ***review_count*** and ***stars*** from the business table, specifically for open restaurant businesses identified earlier. Since SQLite lacks a built-in median function, the query orders the values and selects the middle one to compute the median. The WHERE clause ensures that the results are filtered to only include the businesses that are open restaurants. The results are then transposed, converting rows to columns and vice - versa, making the output easier to read.

Review count and rating together constitute the business score, which is to be analysed for the purpose of this project. The review distribution shows a significant disparity, with a few popular restaurants receiving many reviews, leading to a **positively skewed distribution**. Most restaurants have relatively few reviews. Customer satisfaction appears generally positive, with average and median star ratings around 3.5, indicating satisfactory experiences overall, though there are **outliers**. Overall, the dataset reveals varied customer engagement and satisfaction levels, with a few standout businesses driving most interactions and ratings.

```
# FUNCTION - Outlier Removal using IQR
def remove_outliers(df, col):
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df
```

```
restaurant_business = remove_outliers(restaurant_business,'review_count')
```

```
restaurant_business.shape
```

```
(31537, 2)
```

	0
average_review_count	55.975426
min_review_count	5.000000
max_review_count	248.000000
median_review_count	15.000000
average_star_rating	3.477281
min_star_rating	1.000000
max_star_rating	5.000000
median_star_rating	3.500000

The code defines a function to remove outliers using the IQR method, eliminating data points outside Q1 - 1.5*IQR and Q3 + 1.5*IQR. After applying this to the ***review_count*** column, **31,537** rows remain, with **average review count** being **55**. The final statistics show a normalized distribution of review counts and similar star rating statistics to the original data.

Analysing the relationship between review_count and rating

```
# Identifying the restaurants having the highest number of reviews
pd.read_sql_query(f"""SELECT name, SUM(review_count) as review_count, AVG(stars) AS avg_rating
FROM business
WHERE business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY name
ORDER BY review_count DESC
LIMIT 10;""",conn)
```

	name	review_count	avg_rating
0	McDonald's	16490	1.868702
1	Chipotle Mexican Grill	9071	2.381757
2	Taco Bell	8017	2.141813
3	Chick-fil-A	7687	3.377419
4	First Watch	6761	3.875000
5	Panera Bread	6613	2.661905
6	Buffalo Wild Wings	6483	2.344828
7	Domino's Pizza	6091	2.290210
8	Wendy's	5930	2.030159
9	Chili's	5744	2.514706

The code identifies the top 10 restaurants by review count, with McDonald's topping the list at 16,490 reviews and an average rating of 1.87. Other high-review restaurants include Chipotle Mexican Grill, Taco Bell, and Chick-fil-A, each with varied average ratings.

```
# Identifying the restaurants having the highest number of rating
pd.read_sql_query(f"""SELECT name, SUM(review_count) as review_count, AVG(stars) AS avg_rating
FROM business
WHERE business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY name
ORDER BY avg_rating DESC
LIMIT 10;
""",conn)
```

	name	review_count	avg_rating
0	ā café	48	5.0
1	two birds cafe	77	5.0
2	the brewers cabinet production	13	5.0
3	taqueria la cañada	17	5.0
4	la bamba	44	5.0
5	la 5th av tacos	24	5.0
6	el sabor mexican and chinese food	21	5.0
7	eat.drink.Om...YOGA CAFE	7	5.0
8	d4 Tabletop Gaming Cafe	8	5.0
9	cabbage vegetarian cafe	12	5.0

The code identifies the top 10 restaurants with the highest average rating of 5.0. Their review counts vary between 7 and 77, showing that these restaurants are highly rated, irrespective of the number of reviews.

Output Inference:

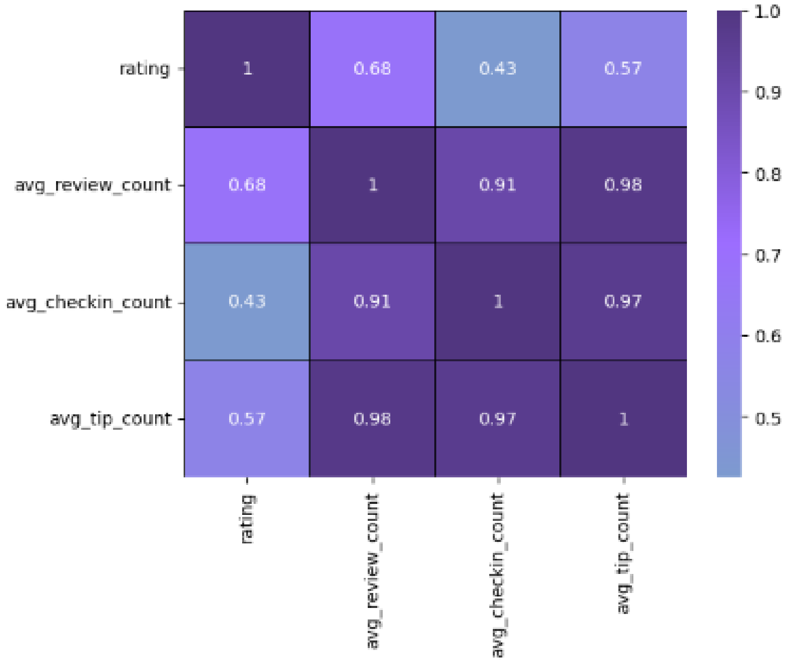
- **No Direct Correlation**: High review counts do not guarantee high ratings, and high ratings do not always correlate with a high number of reviews.
- **Engagement vs. Satisfaction**: Review count indicates user engagement but does not necessarily reflect overall customer satisfaction.
- **Complex Success Factors**: Success in the restaurant business is influenced by multiple factors beyond just review counts and ratings.

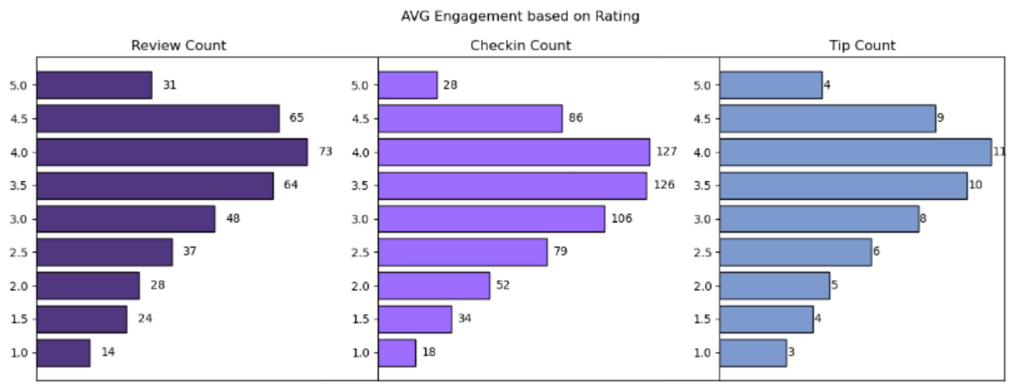
Analysing the relationship between restaurant engagement and ratings

```
review_count_df = pd.read_sql_query(f"""SELECT total.avg_rating as rating,
AVG(total.review_count) as avg_review_count,
AVG(total.checkin_count) as avg_checkin_count,
AVG(total.tip_count) as avg_tip_count
FROM
(SELECT
    b.business_id,
    SUM(b.review_count) AS review_count,
    AVG(b.stars) AS avg_rating,
    SUM(LENGTH(cc.date) - LENGTH(REPLACE(cc.date, ',', '')) + 1) AS checkin_count,
    SUM(tip.tip_count) as tip_count
FROM
    business b
LEFT JOIN
    checkin cc ON b.business_id = cc.business_id
LEFT JOIN
    (select business_id, count(business_id) as tip_count from tip GROUP BY business_id ORDER BY tip_count) as tip on b.business_id = tip.business_id
WHERE b.business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY
    b.business_id) as total

GROUP BY total.avg_rating

""",conn)
display(review_count_df)
```

The code aggregates restaurant data to examine the link between ratings and engagement metrics. It combines review counts, average ratings, check-in counts, and tip counts from multiple tables and groups the results by average rating, revealing how engagement metrics vary with restaurant ratings.

	rating	avg_review_count	avg_checkin_count	avg_tip_count
0	1.0	14.365079	17.518072	2.781513
1	1.5	24.358459	34.480969	3.884654
2	2.0	27.759629	52.386515	4.581058
3	2.5	36.631037	79.349429	6.325225
4	3.0	48.054998	105.970405	8.301950
5	3.5	63.730125	125.781702	10.320786
6	4.0	73.136954	127.139075	11.329362
7	4.5	65.282554	86.177605	8.995201
8	5.0	31.127979	27.545113	4.269082





Output Inference:

- **Higher Ratings Correlate with Higher Engagement Metrics:** Higher-rated restaurants typically have more reviews, check-ins, and tips.
- **Engagement Metrics Vary with Ratings:** Engagement metrics like reviews and check-ins generally increase with higher ratings, peaking at 4 stars.
- **Diminishing Returns at the Top:** Engagement metrics rise with ratings but slow down at the highest rating levels, indicating diminishing returns. This may suggest a saturation point or that only a small, highly satisfied customer base visits these top-rated restaurants.

Examining the Relationship Between Reviews, Tips, and Check-Ins for Businesses

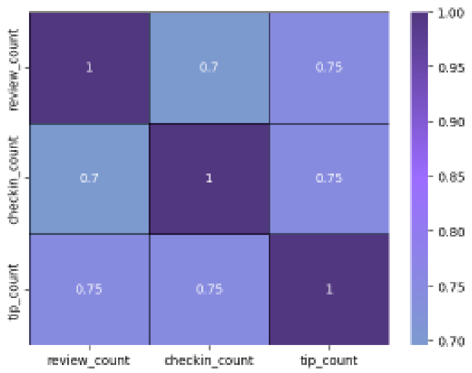
```
engagement_df = pd.read_sql_query(f"""SELECT
    b.business_id,
    SUM(b.review_count) AS review_count,
    AVG(b.stars) AS avg_rating,
    SUM(LENGTH(cc.date) - LENGTH(REPLACE(cc.date, ',', '')) + 1) AS checkin_count,
    SUM(tip.tip_count) as tip_count,
    (CASE WHEN b.stars >= 3.5 THEN 'High-Rated' ELSE 'Low-Rated' END)  as rating_category
FROM
    business b
LEFT JOIN
    checkin cc ON b.business_id = cc.business_id
LEFT JOIN
    (select business_id, count(business_id) as tip_count from tip GROUP BY business_id ORDER BY tip_count) as tip on b.business_id = tip.business_id
WHERE b.business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY
    b.business_id
ORDER BY
    review_count DESC,
    checkin_count DESC;
""",conn).dropna()
```

The above code snippet aggregates data to analyse the relationship between restaurant ratings and user engagement metrics (reviews, check-ins, and tips). It calculates the total number of reviews, average ratings, check-in counts, and tip counts for each restaurant, categorizing them as either 'High-Rated' or 'Low-Rated' based on their average rating (3.5 or above is 'High-Rated'). This allows for a comparison of engagement metrics across different rating categories, showing how user engagement varies between high-rated and low-rated businesses.

	business_id	review_count	avg_rating	checkin_count	tip_count	rating_category
14	30OhTA38fp8xuqW4O2D6Eg	248	4.0	296.0	14.0	High Rated
15	Aw9Tidxcg5ifodzn0R2O6g	248	4.0	252.0	18.0	High Rated
16	9iSoPNBV54dj6L0rxO4RWw	248	3.5	219.0	7.0	High Rated
17	HI1zbZuujFH9yPBKP1GH6g	248	4.5	214.0	21.0	High Rated
18	7dbUShu3yTUVNhTrdnF0FQ	248	4.0	166.0	16.0	High Rated
...	...	...	...	...	...	...
31389	v2xhzKIW-1bySJw5UPy8Jw	5	2.5	1.0	1.0	Low Rated
31392	wp_fwjX8JJCB5F-sgb7ASg	5	5.0	1.0	1.0	High Rated
31393	x3eNFvMD1LaqpBn/SD6A9Q	5	3.0	1.0	1.0	Low Rated
31397	yaJAs2OmRRhsbywHPGMxQ	5	5.0	1.0	3.0	High Rated
31398	z00F0RSAGimvSU9IrTevDw	5	1.0	1.0	1.0	Low Rated

25473 rows × 6 columns

<Axes: >



The heatmap visually represents the correlation matrix for review counts, check-in counts, and tip counts for businesses:

- **Strong Positive Correlations:** The heatmap reveals strong positive correlations between review counts, check-in counts, and tip counts. Specifically, there are correlations of 0.7 between review and check-in counts, 0.75 between review and tip counts, and 0.75 between check-in and tip counts. This indicates that businesses with **higher reviews** tend to have more check-ins and tips as well.
- **Consistent Engagement Across Metrics:** The high correlations suggest a pattern of consistent user engagement, suggesting that successful businesses **attract high** levels of **customer interaction** across multiple engagement metrics, pointing to overall business success.

Comparing User Engagement (Reviews, Tips, and Check-Ins) Between High-Rated and Low-Rated Businesses

```
engagement_df.groupby('rating_category')[['review_count', 'checkin_count', 'tip_count']].mean()
```

rating_category	review_count	checkin_count	tip_count
High-Rated	63.099378	80.71859	8.069794
Low-Rated	37.152862	64.84321	5.456341

The above code groups the ***engagement_df*** by the ***rating_category*** column and calculates the average values for ***review_count***, ***checkin_count***, and ***tip_count*** within each category.

The output shows that high-rated restaurants have higher average review counts (63.10), check-in counts (80.72), and tip counts (8.07) compared to low-rated restaurants, which have average review counts (37.15), check-in counts (64.84), and tip counts (5.46).

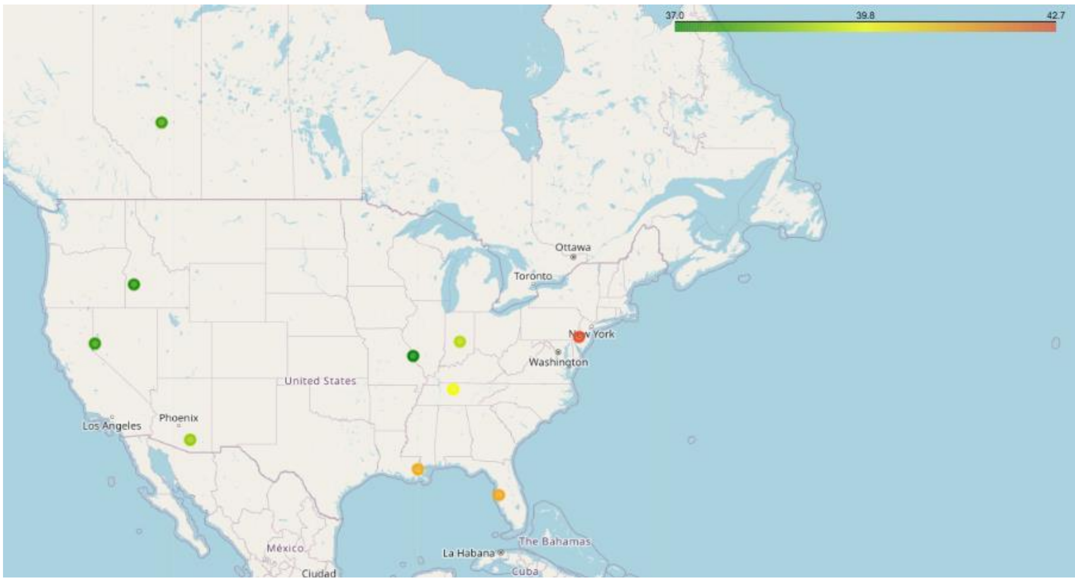
Therefore, it can be inferred that **high-rated restaurants** tend to have **more user engagement** across reviews, check-ins, and tips than low-rated ones, suggesting that better-rated businesses generally attract more customer interactions.

<u>Analysing Variation in Restaurant Success Metrics (Review Count and Average Rating) Across States and Cities.</u>

```python
city_df = pd.read_sql_query(f"""SELECT state,city, latitude, longitude, AVG(stars) AS avg_rating, SUM(review_count) as review_count,
COUNT(*) as restaurant_count
FROM business
WHERE business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY state, city
ORDER BY review_count DESC
limit 10;""",conn)


city_df['success_score'] = calculate_success_metric(city_df)
display(city_df)
```

	state	city	latitude	longitude	avg_rating	review_count	restaurant_count	success_score
0	PA	Philadelphia	39.955505	-75.155564	3.532156	175487	3001	42.651934
1	FL	Tampa	27.890814	-82.502346	3.571429	104376	1715	41.270588
2	IN	Indianapolis	39.637133	-86.127217	3.412111	92639	1701	39.022521
3	AZ	Tucson	32.338572	-111.010760	3.386187	91613	1419	38.688341
4	TN	Nashville	36.208102	-86.768170	3.493590	87070	1404	39.737764
5	LA	New Orleans	29.963974	-90.042604	3.693676	69239	1012	41.167252
6	MO	Saint Louis	38.583223	-90.407187	3.414303	51490	811	37.042331
7	NV	Reno	39.476518	-119.784037	3.479626	48393	589	37.535187
8	AB	Edmonton	53.436403	-113.604288	3.509379	45916	1546	37.671748
9	ID	Boise	43.611192	-116.206275	3.558824	36104	561	37.346958

The following code fetches data about cities, including average ratings and review counts of restaurants. It then calculates a "success score" for each city by multiplying the average rating by the logarithm of the review count plus one and displays the top 10 cities based on this score.

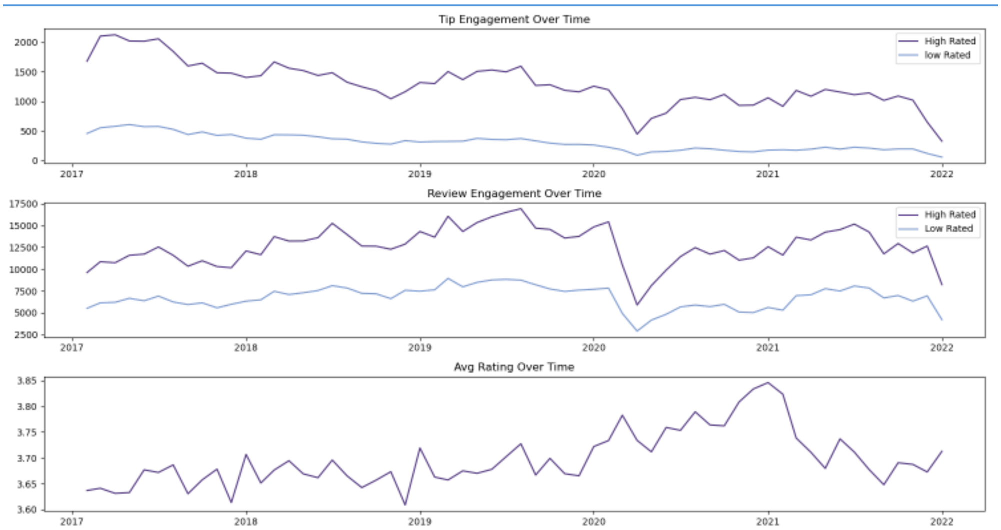


Philadelphia has the highest success score, reflecting a strong **combination** of **high ratings** and **active user engagement**. **Tampa, Indianapolis, and Tucson** also rank highly, indicating they have thriving restaurant scenes with notable success scores. Cities with **higher review counts** and **decent average ratings** generally achieve **higher success scores**, highlighting both quality and popularity in these areas.

<u>Analysing User Engagement Patterns Over Time for Successful vs. Less Successful Businesses</u>

```
high_rated_engagement = pd.read_sql_query(f"""
SELECT review.month_year, review.review_count, tip.tip_count FROM
(SELECT strftime('%m-%Y', date) AS month_year, COUNT(*) AS review_count
FROM review
WHERE business_id IN {tuple(restaurant_business["business_id"])} and stars >= 3.5
GROUP BY month_year
ORDER BY month_year) as review
JOIN
(SELECT AVG(b.stars), strftime('%m-%Y', tip.date) AS month_year, COUNT(*) AS tip_count
FROM tip
JOIN business as b
on tip.business_id = b.business_id
WHERE tip.business_id IN {tuple(restaurant_business['business_id'])} and b.stars >= 3.5
GROUP BY month_year
ORDER BY month_year) as tip

on review.month_year = tip.month_year
;""",conn)

low_rated_engagement = pd.read_sql_query(f"""
SELECT review.month_year, review.review_count, tip.tip_count FROM
(SELECT strftime('%m-%Y', date) AS month_year, COUNT(*) AS review_count
FROM review
WHERE business_id IN {tuple(restaurant_business["business_id"])} and stars < 3.5
GROUP BY month_year
ORDER BY month_year) as review
JOIN
(SELECT AVG(b.stars), strftime('%m-%Y', tip.date) AS month_year, COUNT(*) AS tip_count
FROM tip
JOIN business as b
on tip.business_id = b.business_id
WHERE tip.business_id IN {tuple(restaurant_business['business_id'])} and b.stars < 3.5
GROUP BY month_year
ORDER BY month_year) as tip

on review.month_year = tip.month_year
;""",conn)

time_rating = pd.read_sql_query(f"""SELECT strftime('%m-%Y', date) AS month_year, AVG(stars) as avg_rating
FROM review
WHERE business_id IN {tuple(restaurant_business["business_id"])}
GROUP BY month_year
ORDER BY month_year
;""",conn)
time_rating['month_year'] = pd.to_datetime(time_rating['month_year'])
time_rating.sort_values('month_year',inplace = True)
time_rating = time_rating[time_rating['month_year']>'2017']

high_rated_engagement['month_year'] = pd.to_datetime(high_rated_engagement['month_year'])
high_rated_engagement.sort_values('month_year',inplace = True)
high_rated_engagement = high_rated_engagement[high_rated_engagement['month_year']>'2017']

low_rated_engagement['month_year'] = pd.to_datetime(low_rated_engagement['month_year'])
low_rated_engagement.sort_values('month_year',inplace = True)
low_rated_engagement = low_rated_engagement[low_rated_engagement['month_year']>'2017']
high_rated_engagement['avg_rating'] = time_rating['avg_rating'].values
```

The code generates three data frames to analyse user engagement trends over time for restaurants based on their ratings. The ***high_rated_engagement*** data frame captures the monthly counts of reviews and tips for restaurants with ratings of 3.5 stars or higher, while the ***low_rated_engagement*** data frame captures the same metrics for restaurants with ratings below 3.5 stars. Additionally, the ***time_rating*** data frame tracks the average monthly rating of all restaurants since 2017. These data frames facilitate a time series analysis to observe and compare patterns in user engagement and ratings over time for high-rated and low-rated restaurants.



The three time series plots provide insights into user engagement and average ratings over time for high-rated and low-rated restaurants.

- Tip Engagement Over Time: Indicates that **high-rated** restaurants consistently receive **more tips**, exhibiting a general stability overall compared to low-rated ones, which mostly have minimal engagement. There is a noticeable **drop** in tip engagement around **early 2020**, likely due to the **COVID-19 pandemic**, affecting both high-rated and low-rated restaurants.
- Review Engagement Over Time: Reveals that **high-rated** restaurants consistently receive **more reviews** compared to their lower-rated counterparts. This pattern suggests that higher-rated restaurants maintain a **steady or increasing level of user engagement** over time, reflecting **sustained customer interest** and **satisfaction**. The **dip** in review engagement around **early 2020** aligns with the **pandemic**, but **high-rated** restaurants **recover faster** than low-rated ones, indicating resilience and ongoing customer interaction.
- Avg Rating Over Time: Indicates the average rating of all restaurants over the years. The average rating remains **relatively stable** with minor fluctuations, indicating that overall customer satisfaction levels have not significantly changed over time. **High-rated** restaurants **consistently maintain higher average ratings** compared to low-rated ones, reinforcing their superior customer satisfaction and engagement levels.

Identifying Seasonal Trends in User Engagement for Restaurants

```
tip_high_rated = high_rated_engagement[['month_year','tip_count']].set_index('month_year')
review_high_rated = high_rated_engagement[['month_year','review_count']].set_index('month_year')
rating_df = time_rating[['month_year','avg_rating']].set_index('month_year')
```

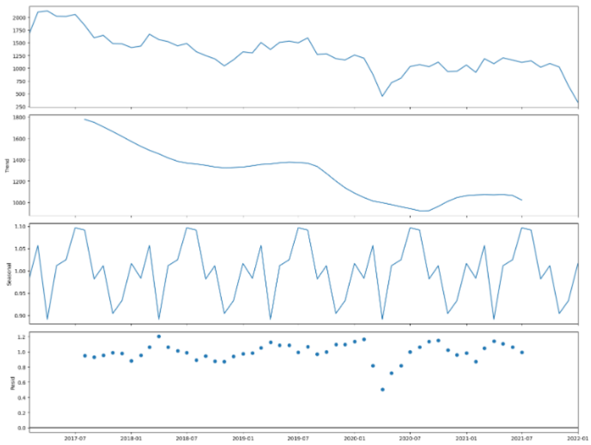
The above code snippet prepares the following DataFrames:

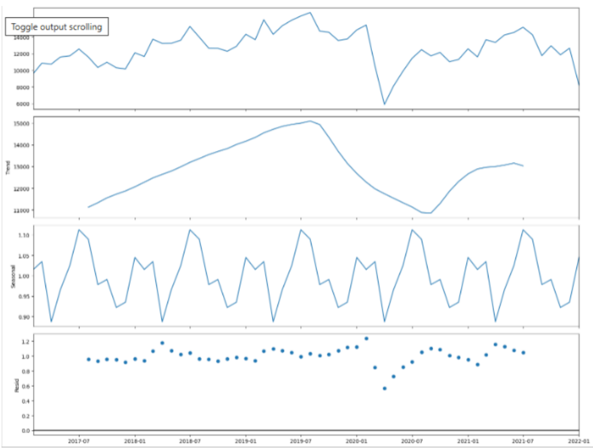
- **tip_high_rated**: Contains the tip counts over time for high-rated restaurants.
- **review_high_rated**: Contains the review counts over time for high-rated restaurants.
- **rating_df**: Contains the average ratings over time for all restaurants.

```
# Seasonal Trends for Tip Count
from statsmodels.tsa.seasonal import seasonal_decompose
multiplicative_decomposition = seasonal_decompose(tip_high_rated,
                                                  model='multiplicative', period = 12)
plt.rcParams.update({'figure.figsize': (16,12)})
multiplicative_decomposition.plot()
plt.show()
```

```
# Seasonal Trends for Review Count
multiplicative_decomposition = seasonal_decompose(review_high_rated,
                                                  model='multiplicative', period = 12)
plt.rcParams.update({'figure.figsize': (16,12)})
multiplicative_decomposition.plot()
plt.show()
```

The above code performs and plots a multiplicative seasonal decomposition of tip counts and review counts for high-rated restaurants. It uses the ***seasonal_decompose*** function to break down the data into trend, seasonal, and residual components, and then plots these components to visualize the underlying patterns.



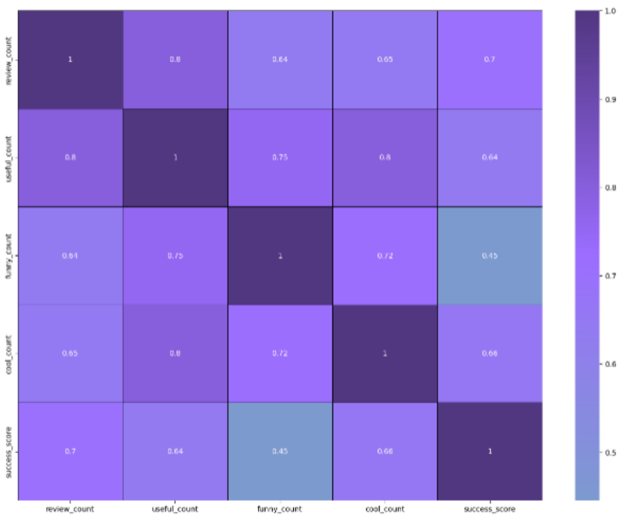


The output clearly indicates that the **tip counts** show a **downward trend** over time, while **review counts** exhibit an **upward trend**. Engagement is particularly high and seasonal from around **November to March** each year.

Correlating Sentiment of Reviews and Tips with Restaurant Success Metrics

```
sentiment_df = pd.read_sql_query(f"""SELECT b.business_id, AVG(b.stars) as avg_rating, SUM(b.review_count) as review_count,
SUM(s.useful_count) as useful_count,
SUM(s.funny_count) as funny_count,
SUM(s.cool_count) as cool_count
FROM
(SELECT business_id,
    SUM(useful) as useful_count,
    SUM(funny) as funny_count,
    SUM(cool) as cool_count
FROM
    review
GROUP BY business_id) as s
JOIN business as b on b.business_id = s.business_id

WHERE b.business_id IN {tuple(restaurant_business['business_id'])}
GROUP BY b.business_id
ORDER BY review_count""",conn)

sentiment_df = remove_outliers(sentiment_df,'review_count')
sentiment_df = remove_outliers(sentiment_df,'useful_count')
sentiment_df = remove_outliers(sentiment_df,'funny_count')
sentiment_df = remove_outliers(sentiment_df,'cool_count')

sentiment_df['success_score'] = calculate_success_metric(sentiment_df)

sns.heatmap(sentiment_df.iloc[:,2:].corr(), cmap = custom_cmap, annot = True, linewidths=0.5, linecolor = 'black')

plt.show()
```

This code retrieves sentiment data for restaurants, processes it, calculates a success score, and visualizes correlations. It queries a database to get average ratings, review counts, and sentiment counts (useful, funny, cool) for each business. The data is then cleaned by removing outliers in these counts and a success score is calculated for each business.



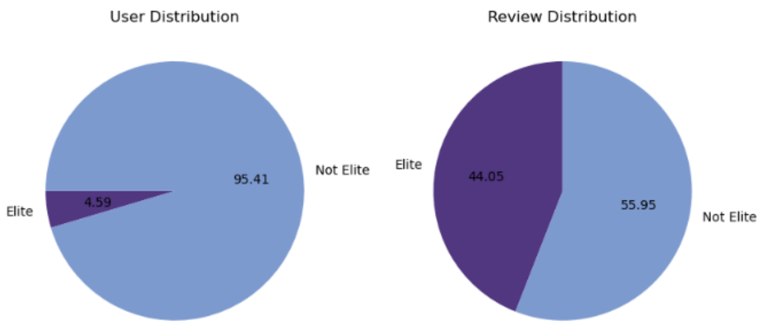
The heatmap reveals **strong positive correlations** between review count, useful votes, and success score, indicating that more reviews and higher useful counts significantly contribute to a restaurant's success. Funny reviews are often seen as useful, and cool reviews are moderately correlated with other attributes. Overall, **higher engagement** through **reviews** and **positive feedback impacts** a restaurant's **success**.

Comparing Engagement Levels Between Elite and Non-Elite Users

```
elite_df = pd.read_sql_query("""SELECT
    elite,
    COUNT(*) AS row_count,
    SUM(review_count) AS total_review_count
FROM
    (SELECT
        CASE
            WHEN elite = '' THEN 'Not Elite'
            ELSE 'Elite'
        END AS elite,
        u.review_count
    FROM
        user u) AS user_elite
GROUP BY
    elite;
""",conn)
elite_df
```

This code retrieves and processes data about elite and non-elite users from a database. It classifies users as "Elite" or "Not Elite" based on their elite status, counts the number of users in each group, and sums their total review counts.

	elite	row_count	total_review_count
0	Elite	91198	20484441
1	Not Elite	1896699	26021235



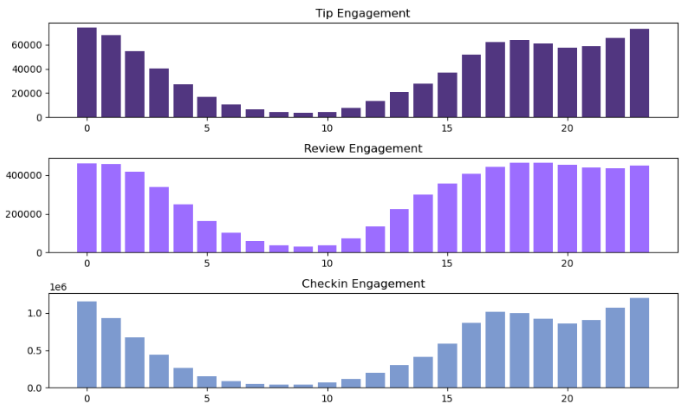
The outputs indicate that **elite users**, though fewer in number (91,198), contribute a **substantial proportion of reviews** (20,484,441) compared to non-elite users (1,896,699 users with 26,021,235 reviews). Establishing a **positive relationship** with elite users can lead to **repeat**

visits and loyalty, as they are more likely to continue supporting businesses, they have had good experiences with, thereby **benefiting businesses**.

Identifying the Busiest Hours for Restaurants

```
review_engagement = pd.read_sql_query("""SELECT
  cast (strftime('%H',date) as integer)
  as hour,
  COUNT(*) AS review_count
FROM
  review
GROUP BY
  hour;
""",conn)

tip_engagement = pd.read_sql_query("""SELECT
  cast (strftime('%H',date) as integer)
  as hour,
  COUNT(*) AS tip_count
FROM
  tip
GROUP BY
  hour;
""",conn)

checkin =  pd.read_sql_query("""SELECT date FROM checkin""",conn)
checkin_engagement = []
for i in checkin['date']:
    checkin_engagement.extend([datetime.strptime(j.strip(),"%Y-%m-%d %H:%M:%S").strftime("%H") for j in i.split(',')])

checkin_engagement = pd.DataFrame(checkin_engagement).astype('int').groupby(0)[[0]].count()
```

The code collects and processes data on user interactions with a service. It retrieves the number of reviews and tips submitted each hour and counts the number of check-ins for each hour. The final output includes hourly engagement metrics for reviews, tips, and check-ins, which can be used to analyse patterns in user activity throughout the day.



The data indicates that the busiest hours for restaurants, in terms of user engagement, range from **4 pm to 1 am**. Understanding these **peak** times enables businesses to **adjust** their **staffing and resources** effectively, ensuring **smooth operations** and **high-quality service**. The **elevated** engagement during the **evening** and **night hours** suggests a **greater demand** for dining out, likely influenced by factors such as **work schedules, social events,** and **recreational activities**.

Recommendation

- By **analysing metrics** like user engagement, review sentiment, peak hours, and the role of elite users, businesses can make **strategic decisions to drive success**.
- It's essential to **understand customer preferences, behaviour, and satisfaction**. Focusing on exceptional experiences will help meet and exceed customer expectations.
- Leveraging data on peak hours and engagement allows businesses to **optimize** staffing, resource allocation, and operating hours, ensuring **efficient and high-quality service** during **busy times**.
- Positive reviews from elite users and high engagement can **enhance** a business's **online presence** and **reputation**. Engaging actively with customers and responding quickly to feedback are key to building credibility and attracting new clients.
- **Collaborating** with influential **elite** users can boost promotional efforts, increase brand awareness, and drive customer growth. Developing strong relationships with loyal customers can also reinforce a business's market position.
- Adjusting operating hours or offering special promotions can help **capitalize** on the increased **demand** during peak times.
- Businesses struggling with lower success scores should focus on **improving user engagement** by enhancing service quality and responding to customer feedback.
- Cities with **high success scores** offer opportunities for restaurant chains to **expand** or **invest** further.

REFERENCES

https://www.lifewire.com/what-is-yelp-4685842