Importing Libraries

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import numpy as np
import sqlite3
import folium
from geopy.geocoders import Nominatim
from matplotlib.colors import LinearSegmentedColormap
from IPython.display import display
import warnings
warnings.filterwarnings('ignore')
```

Database connection

In [3]:
```
# creating database connection
conn = sqlite3.connect('yelp.db')
```

In [4]:
```
tables = pd.read_sql_query("""SELECT name FROM sqlite_master WHERE type = 'table'""
tables
```

Out[4]:

	name
0	business
1	review
2	user
3	tip
4	checkin

In [5]:
```
# explore what type of data is available in the tables
for table in tables['name']:
    display(pd.read_sql_query(f"SELECT * FROM {table} LIMIT 5",conn))
```

	business_id	name	address	city	state	postal_code	latitude	
0	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA	93101	34.426679	-
1	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO	63123	38.551126	
2	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ	85711	32.223236	-
3	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA	19107	39.955505	
4	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA	18054	40.338183	

	review_id	user_id	business_id	stars	useful	f
0	KU_O5udG6zpxOg-VcAEodg	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3.0	0	
1	BiTunyQ73aT9WBnpR9DZGw	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	5.0	1	
2	saUsX_uimxRlCVr67Z4Jig	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3.0	0	
3	AqPFMleE6RsU23_auESxiA	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5.0	1	
4	Sx8TMOWLNuJBWer-0pcmoA	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	4.0	1	

	user_id	name	review_count	yelping_since	useful	funny	cool	
0	qVc8ODYU5SZjKXVBgXdI7w	Walker	585	2007-01-25 16:47:26	7217	1259	5994	
1	j14WgRoU_-2ZE1aw1dXrJg	Daniel	4333	2009-01-25 04:35:42	43091	13066	27281	2009,201(
2	2WnXYQFK0hXEoTxPtV2zvg	Steph	665	2008-07-25 10:41:00	2086	1010	1003	
3	SZDeASXq7o05mMNLshsdIA	Gwen	224	2005-11-29 04:38:33	512	330	299	
4	hA5lMy-EnncsH4JoR-hFGQ	Karen	79	2007-01-05 19:40:59	29	15	7	

5 rows × 22 columns

	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	QoezRbYQncpRqyrLH6Iqjg	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	ld0AperBXk1h6UbqmM80zw	_uN0OudeJ3Zl_tf6nxg5ww	Appetizers.. platter special for lunch	2012-10-06 19:43:09	0

	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...

Data Analysis

In [6]:
```
pd.read_sql_query("SELECT COUNT(*) FROM business",conn)
```

Out[6]:

	COUNT(*)
0	150346

In [7]:
```
business_id = pd.read_sql_query(""" SELECT business_id,review_count FROM business W
```

In [8]:
```
# What is the descriptive stats for review count and star rating for businesses?
# average, min, max, median

pd.read_sql_query(f"""
SELECT AVG(review_count),
       MIN(review_count),
       MAX(review_count),
       (SELECT review_count FROM business ORDER BY review_count LIMIT 1 OFFSET (SEL
       FROM business WHERE business_id IN {tuple(business_id['business_id'])}""",cc
```

Out[8]:

	AVG(review_count)	MIN(review_count)	MAX(review_count)	median_review_count
0	104.097789	5	7568	15

In [9]:
```
# What is the description stats for review count and star rating for businesses?
# avg, min, max, median
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 COU

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 bus

FROM business
WHERE business_id IN {tuple(business_id['business_id'])};""",conn).transpose()
```

Out[9]:

	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

In [10]:
```
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
```

In [11]:
```
business_id = remove_outliers(business_id,'review_count')
```

In [12]:
```
business_id.shape
```

Out[12]:
```
(31537, 2)
```

In [13]:
```
# What is the description stats for review count and star rating for businesses?
# avg, min, max, median
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 COU

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 bus

FROM business
WHERE business_id IN {tuple(business_id['business_id'])};""",conn).transpose()
```

Out[13]:

	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

In [49]:

```
# Which restaurants have 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(business_id['business_id'])}
GROUP BY name
ORDER BY review_count DESC
LIMIT 10""",conn)
```

Out[49]:

	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

In [15]:

```
# Do restaurants with higher engagement tend to have higher ratings?
pd.read_sql_query("""
SELECT business_id,
SUM(length(date) - length(replace(date,',',''))+1) AS checkin_count
FROM checkin
GROUP BY business_id""",conn)
```

Out[15]:

	business_id	checkin_count
0	---kPU91CF4Lq2-WlRu9Lw	11
1	--0iUa4sNDFiZFrAdIWhZQ	10
2	--30_8IhuyMHbSOcNWd6DQ	2
3	--7PUidqRWpRSpXebiyxTg	10
4	--7jw19RH9JKXgFohspgQw	26
...	...	...
131925	zznJox6-nmXlGYNWgTDwQQ	67
131926	zznZqH9CiAznbkV6fXyHWA	1
131927	zzu6_r3DxBJuXcjnOYVdTw	23
131928	zzw66H6hVjXQEt0Js3Mo4A	2
131929	zzyx5x0Z7xXWWvWnZFuxlQ	1

131930 rows × 2 columns

In [16]:
```
pd.read_sql_query("""SELECT business_id, COUNT(*) AS tip_count
FROM tip
GROUP BY business_id""",conn)
```

Out[16]:

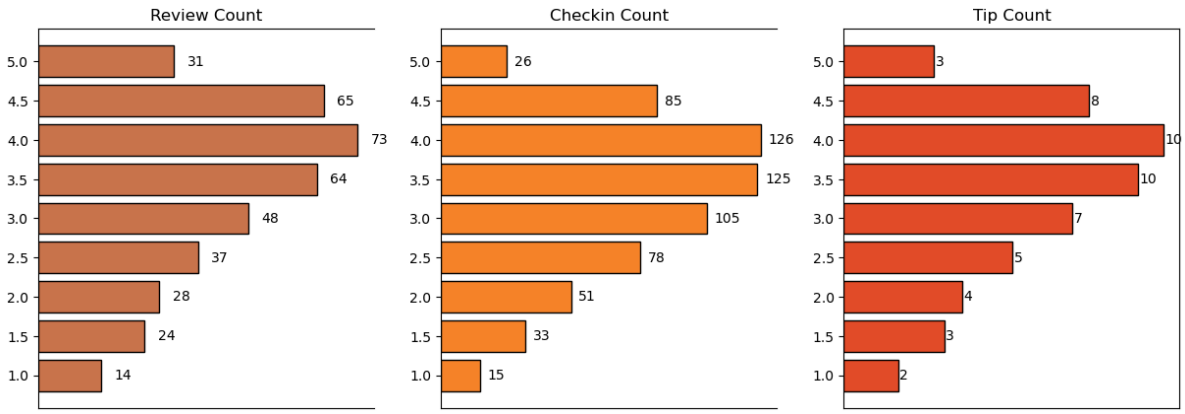
	business_id	tip_count
0	---kPU91CF4Lq2-WlRu9Lw	4
1	--0iUa4sNDFiZFrAdIWhZQ	6
2	--30_8IhuyMHbSOcNWd6DQ	1
3	--7PUidqRWpRSpXebiyxTg	3
4	--8IbOsAAxjKRoYsBFL-PA	4
...	...	...
106188	zzjCxn89a7RQo8kelOO_Ag	1
106189	zzjFdJwXuxBOGe9JeY_EMw	2
106190	zznJox6-nmXlGYNWgTDwQQ	6
106191	zzu6_r3DxBJuXcjnOYVdTw	2
106192	zzyx5x0Z7xXWWvWnZFuxlQ	2

106193 rows × 2 columns

In [17]:
```
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,
```

```
        IFNULL(SUM(LENGTH(cc.date) - LENGTH(REPLACE(cc.date, ',', '')) + 1), 0) AS
        IFNULL(SUM(tip.tip_count), 0) 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
        ) AS tip ON b.business_id = tip.business_id
    WHERE b.business_id IN {tuple(business_id['business_id'])}
    GROUP BY b.business_id
) AS total
GROUP BY total.avg_rating;
""",conn)
```

In [18]:
```
plt.figure(figsize = (15,5))
plt.title('AVG Engagement based on Rating\n\n')
plt.yticks([])
plt.xticks([])
plt.subplot(1,3,1)
plt.title('Review Count')
plt.barh(review_count_df['rating'].astype('str'),review_count_df['avg_review_count'
plt.gca().spines['right'].set_visible(False)
for i, value in enumerate(review_count_df['avg_review_count']):
    plt.text(value+3,i,str(round(value)),color = 'black',va = 'center')

plt.xticks([])
plt.subplot(1,3,2)
plt.title('Checkin Count')
plt.barh(review_count_df['rating'].astype('str'),review_count_df['avg_checkin_count
plt.gca().spines['right'].set_visible(False)
for i,value in enumerate(review_count_df['avg_checkin_count']):
    plt.text(value+3,i,str(round(value)),color='black',va='center')

plt.xticks([])
plt.subplot(1,3,3)
plt.title('Tip Count')
plt.barh(review_count_df['rating'].astype('str'),review_count_df['avg_tip_count'],e
for i, value in enumerate(review_count_df['avg_tip_count']):
    plt.text(value+0.05,i,str(round(value)),color='black',va = 'center')
plt.xticks([])
plt.show()
```



In [19]:
```
# Is there a correlation between the number of reviews, tips, and check-ins for a b
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_cou
        SUM(tip.tip_count) AS tip_count,
        (CASE WHEN b.stars>=3.5 THEN 'High-Rated' ELSE 'Low-Rated' END) AS 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
        ) AS tip ON b.business_id = tip.business_id
    WHERE b.business_id IN {tuple(business_id['business_id'])}
    GROUP BY b.business_id

""",conn).dropna()
```

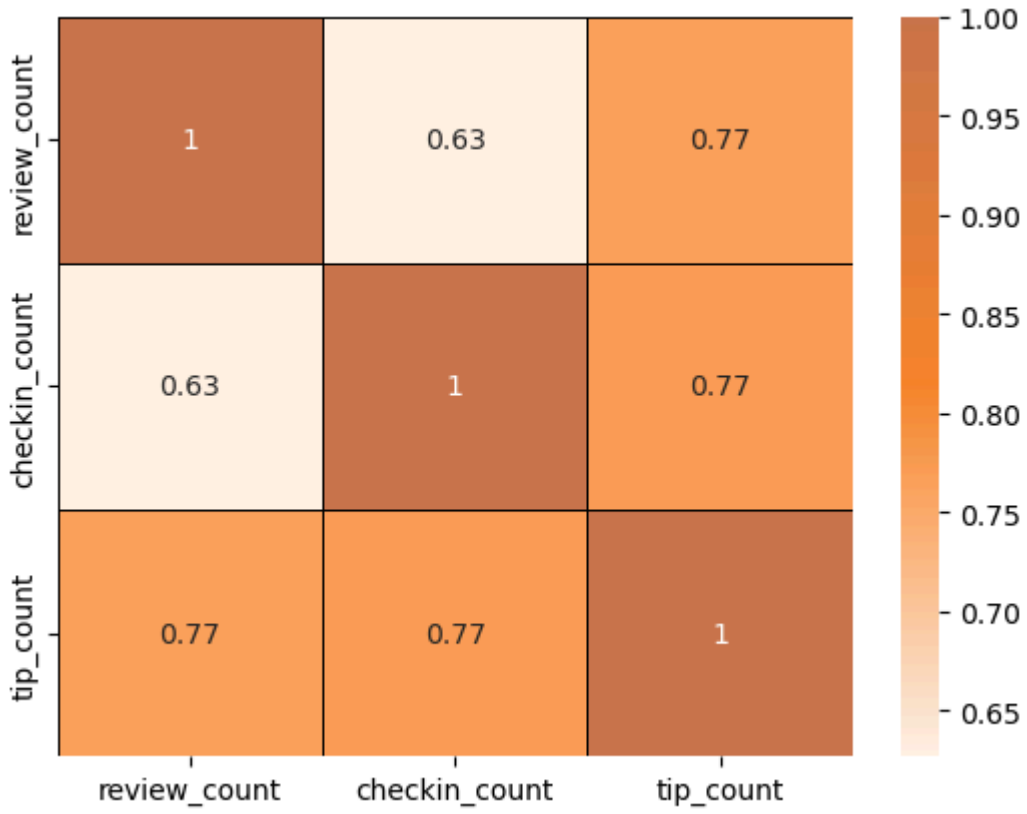
In [20]:
```
engagement_df[['review_count','checkin_count','tip_count']].corr()
```

Out[20]:

	review_count	checkin_count	tip_count
review_count	1.000000	0.626884	0.766321
checkin_count	0.626884	1.000000	0.772088
tip_count	0.766321	0.772088	1.000000

In [21]:
```
colors = ['#FFF1E5','#F8862C','#CB754B']
custom_cmap = LinearSegmentedColormap.from_list("mycmap",colors)
sns.heatmap(engagement_df[['review_count','checkin_count','tip_count']].corr(),cmap
```

Out[21]:
```
<Axes: >
```



In [22]:
```
# Is there a difference in the user engagement (reviews, tips and check-ins) betwee
# Rating more than 3.5 is high-rated, less than 3.5 means low rated

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 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
WHERE b.business_id IN {tuple(business_id['business_id'])}
GROUP BY b.business_id""",conn).dropna()
```

Out[22]:

	business_id	review_count	avg_rating	checkin_count	tip_count	category
0	---kPU91CF4Lq2-WlRu9Lw	24	4.5	11.0	4.0	High-Rated
1	--0iUa4sNDFiZFrAdIWhZQ	14	3.0	10.0	6.0	Low-Rated
2	--epgcb7xHGuJ-4PUeSLAw	34	3.0	118.0	6.0	Low-Rated
4	--lqIzK-ZVTtgwiQM63XgQ	15	2.0	21.0	2.0	Low-Rated
5	-09Oc2D14vRnmirPh0vlXw	135	3.0	500.0	25.0	Low-Rated
...	...	...	...	...	...	...
31530	zzlF9qp2UoHN48EeZH_IDg	19	3.0	6.0	5.0	Low-Rated
31532	zzbZtgPYZS8sTIWQH6DwEw	86	3.0	292.0	17.0	Low-Rated
31533	zziDpuuJw-Km1J4BaGpBKA	6	3.5	20.0	2.0	High-Rated
31534	zzjFdJwXuxBOGe9JeY_EMw	47	4.0	27.0	2.0	High-Rated
31535	zznJox6-nmXlGYNWgTDwQQ	30	1.5	67.0	6.0	Low-Rated

27662 rows × 6 columns

In [23]:
```
# Is there a difference in the user engagement (reviews, tips and check-ins) betwee
# Rating more than 3.5 is high-rated, less than 3.5 means low rated
engagement_df.groupby('category')[['review_count','tip_count','checkin_count']].mea
```

Out[23]:

category	review_count	tip_count	checkin_count
High-Rated	72.291062	10.162766	122.066641
Low-Rated	42.123420	6.541689	88.880828

In [24]:
```
# function to calculate the success score based on the avg rating and total review
def calculate_success_metric(df):
    success_score = []
    for index, row in df.iterrows():
        score = row['avg_rating']*np.log(row['review_count']+1)
        success_score.append(score)
    return success_score
```

In [25]:
```
# How do the success metrics (review_count or avg_rating) of restaurants vary acros
city_df =  pd.read_sql_query(f"""SELECT city,state,latitude,longitude, AVG(stars) A
COUNT(*) AS restaurant_count
FROM business
WHERE business_id IN {tuple(business_id['business_id'])}
GROUP BY state, city
ORDER BY review_count DESC
```

```
LIMIT 10""",conn)

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

In [26]:

```
# Create a base map
m = folium.Map(location = [city_df['latitude'].mean(),city_df['longitude'].mean()],

# Define a color scale
color_scale = folium.LinearColormap(colors = ['green','yellow','#E54F29'],
                                    vmin = city_df['success_score'].min(),
                                    vmax = city_df['success_score'].max())

# Add markers to the map
for index, row in city_df.iterrows():
    folium.CircleMarker(
    location = [row['latitude'],row['longitude']],
    radius = 5,
    color = color_scale(row['success_score']),
    fill=True,
    fill_color = color_scale(row['success_score']),
    fill_opacity = 0.7,
    popup = f"Success Score:{row['success_score']}").add_to(m)

# Add color scale to the map
m.add_child(color_scale)
```

Out[26]:

In [27]:

```
# Are there any seasonal trends in the user engagement for restaurants?
# Are there any patterns in user engagement over time for successful businesses com
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(business_id['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(business_id['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(business_id['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(business_id['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)
```

In [28]:
```
high_rated_engagement
```

Out[28]:

	month_year	review_count	tip_count
0	01-2010	1218	79
1	01-2011	2171	621
2	01-2012	3086	1321
3	01-2013	3801	1230
4	01-2014	4973	1357
...	...	...	...
149	12-2017	10161	1477
150	12-2018	12870	1163
151	12-2019	13756	1161
152	12-2020	11294	937
153	12-2021	12652	652

154 rows × 3 columns

In [29]:
```
low_rated_engagement
```

Out[29]:

	month_year	review_count	tip_count
0	01-2010	613	25
1	01-2011	1103	297
2	01-2012	1748	538
3	01-2013	2196	548
4	01-2014	2769	607
...	...	...	...
149	12-2017	5970	441
150	12-2018	7574	338
151	12-2019	7591	275
152	12-2020	5014	148
153	12-2021	6937	122

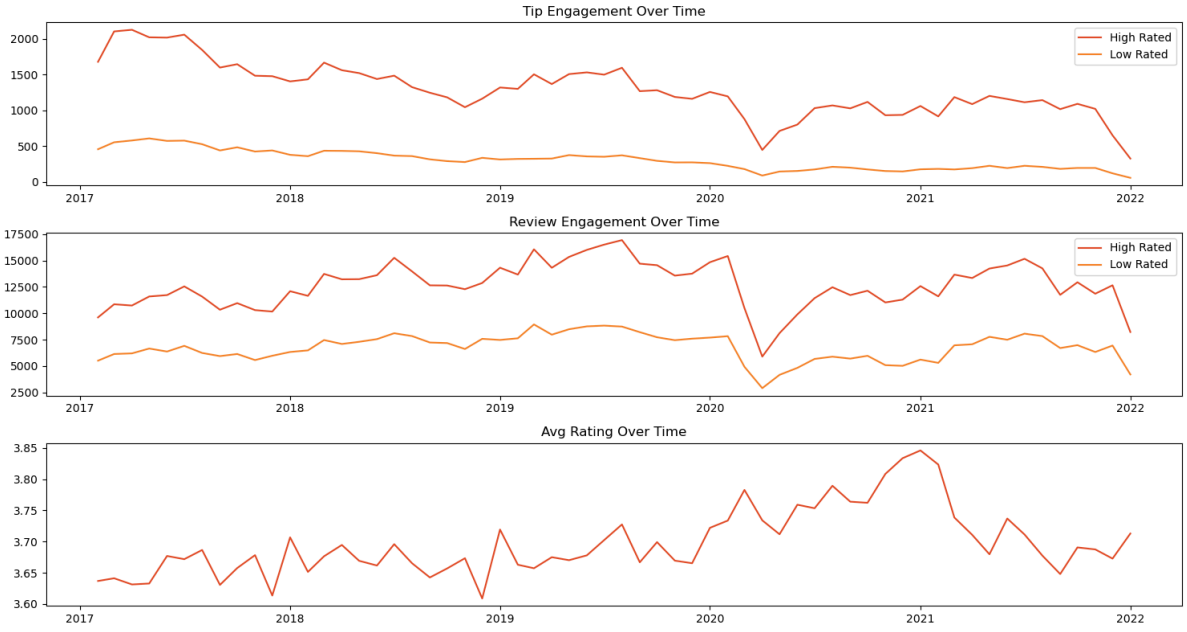
154 rows × 3 columns

In [30]:
```
time_rating = pd.read_sql_query(f"""SELECT strftime('%m-%Y',date) AS month_year,AVG
FROM review
WHERE business_id IN {tuple(business_id['business_id'])}
GROUP BY month_year
ORDER BY month_year;""",conn)
```

In [31]:
```
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_y
high_rated_engagement.sort_values('month_year',inplace = True)
high_rated_engagement = high_rated_engagement[high_rated_engagement['month_year']>'

low_rated_engagement['month_year'] = pd.to_datetime(low_rated_engagement['month_yea
low_rated_engagement.sort_values('month_year',inplace = True)
low_rated_engagement = low_rated_engagement[low_rated_engagement['month_year']>'201
```

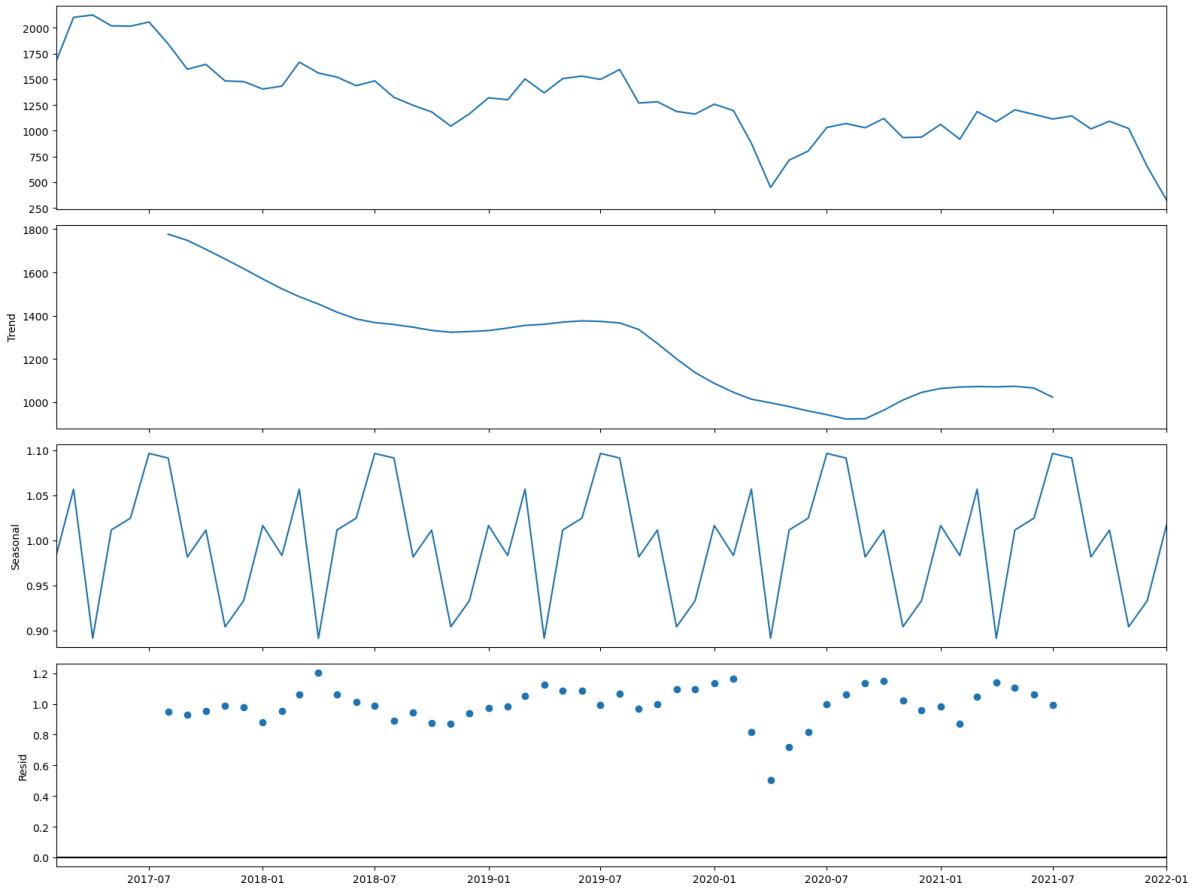
In [32]:
```
high_rated_engagement['avg_rating'] = time_rating['avg_rating'].values
```

In [33]:
```
plt.figure(figsize = (15,8))
plt.subplot(3,1,1)
plt.title('Tip Engagement Over Time')
plt.plot(high_rated_engagement['month_year'],high_rated_engagement['tip_count'],lab
plt.plot(low_rated_engagement['month_year'],low_rated_engagement['tip_count'],label
plt.legend()
plt.subplot(3,1,2)
plt.title('Review Engagement Over Time')
plt.plot(high_rated_engagement['month_year'],high_rated_engagement['review_count'],
plt.plot(low_rated_engagement['month_year'],low_rated_engagement['review_count'],la
plt.legend()
plt.subplot(3,1,3)
plt.title('Avg Rating Over Time')
plt.plot(time_rating['month_year'],time_rating['avg_rating'],color = '#E54F29')
plt.tight_layout()
plt.show()
```



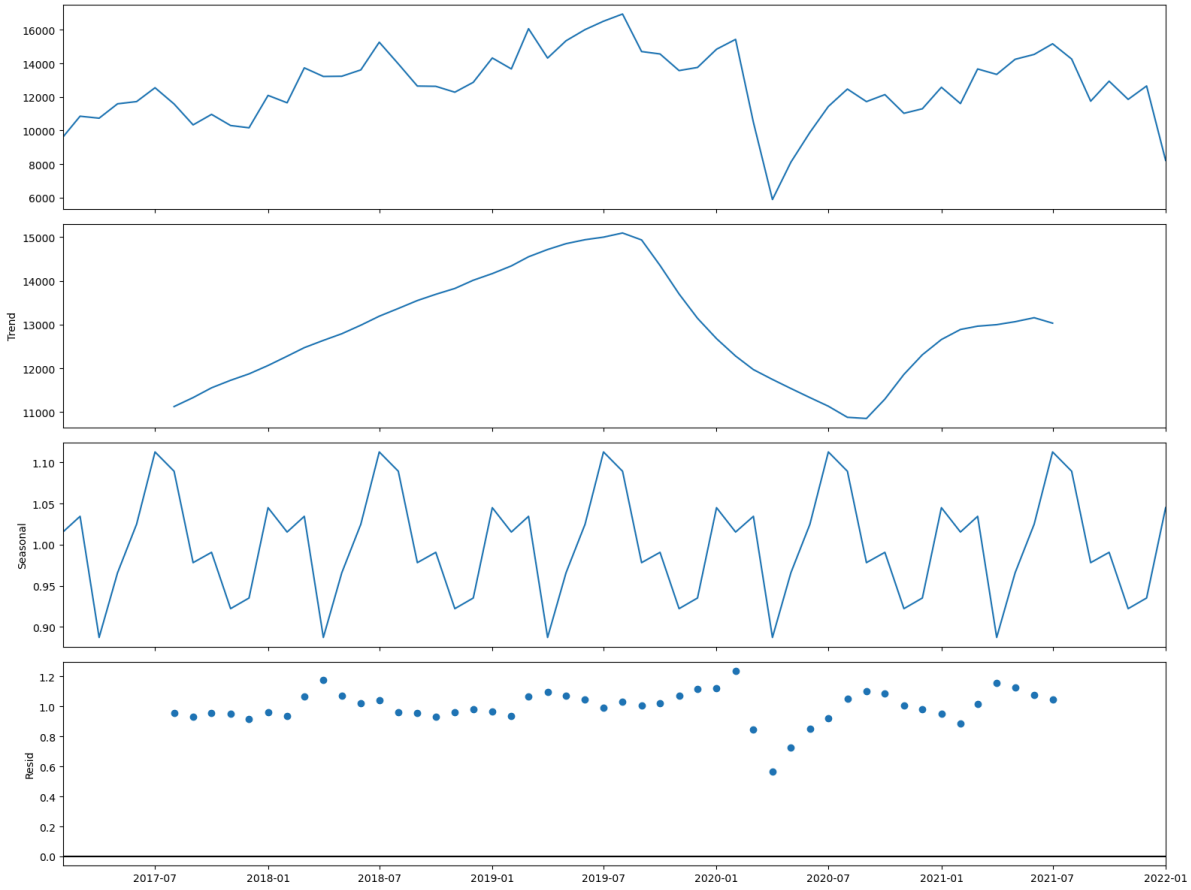
In [34]:
```
tip_high_rated = high_rated_engagement[['month_year','tip_count']].set_index('month
review_high_rated = high_rated_engagement[['month_year','review_count']].set_index(
rating_df = time_rating[['month_year','avg_rating']].set_index('month_year')
```

In [35]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
multiplicative_decomposition = seasonal_decompose(tip_high_rated,model = 'multiplic

plt.rcParams.update({'figure.figsize':(16,12)})
multiplicative_decomposition.plot()
plt.show()
```



In [36]:
```
multiplicative_decomposition = seasonal_decompose(review_high_rated,model = 'multip

plt.rcParams.update({'figure.figsize':(16,12)})
```

```
multiplicative_decomposition.plot()
plt.show()
```

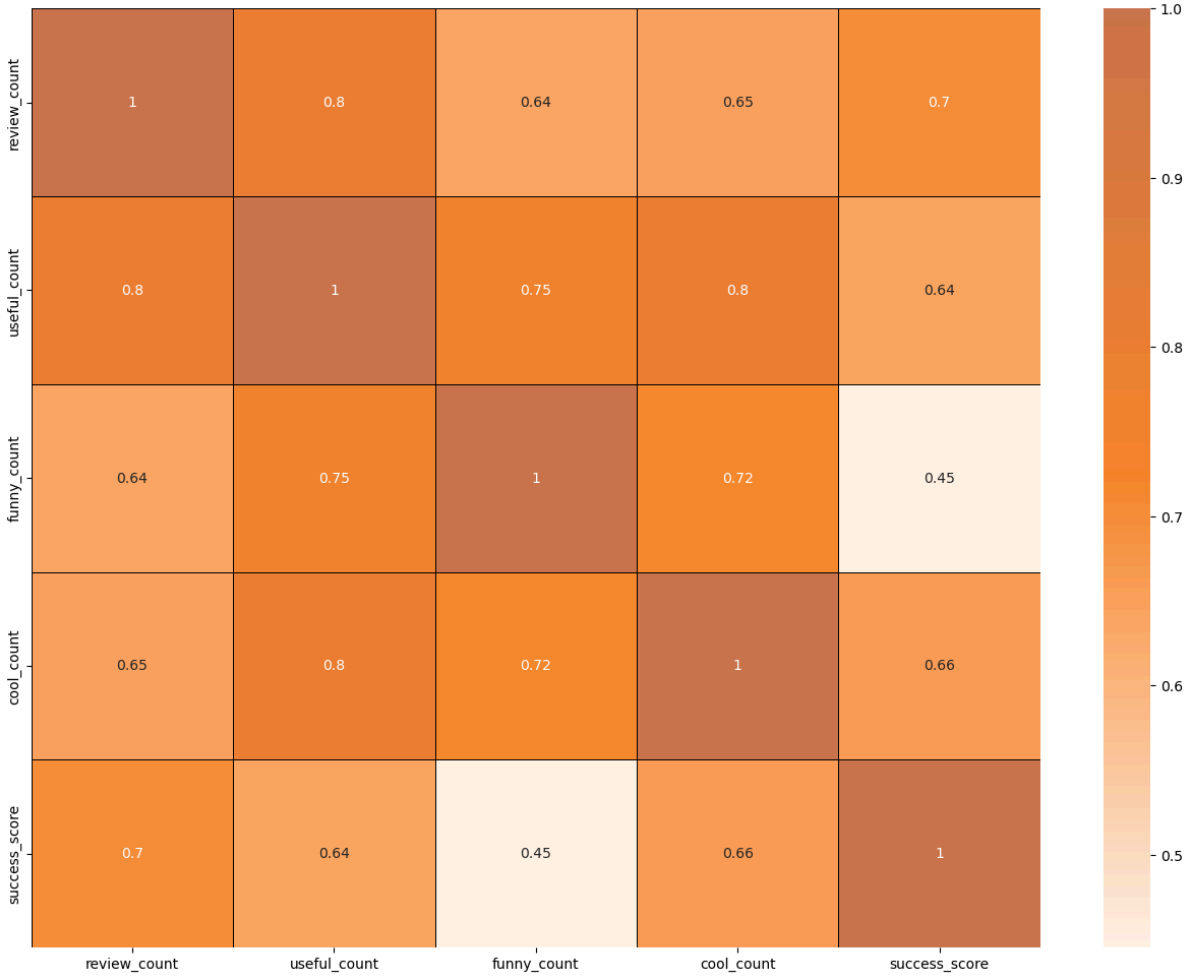


In [37]:
```
# How does the sentiment of reviews and tips (useful, funny, cool) correlate with t
```

In [38]:
```
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(business_id['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')
```

In [39]:
```
sentiment_df['success_score'] = calculate_success_metric(sentiment_df)
```

In [40]:
```
sns.heatmap(sentiment_df.iloc[:,2:].corr(),cmap = custom_cmap,annot = True,linewidt
plt.show()
```



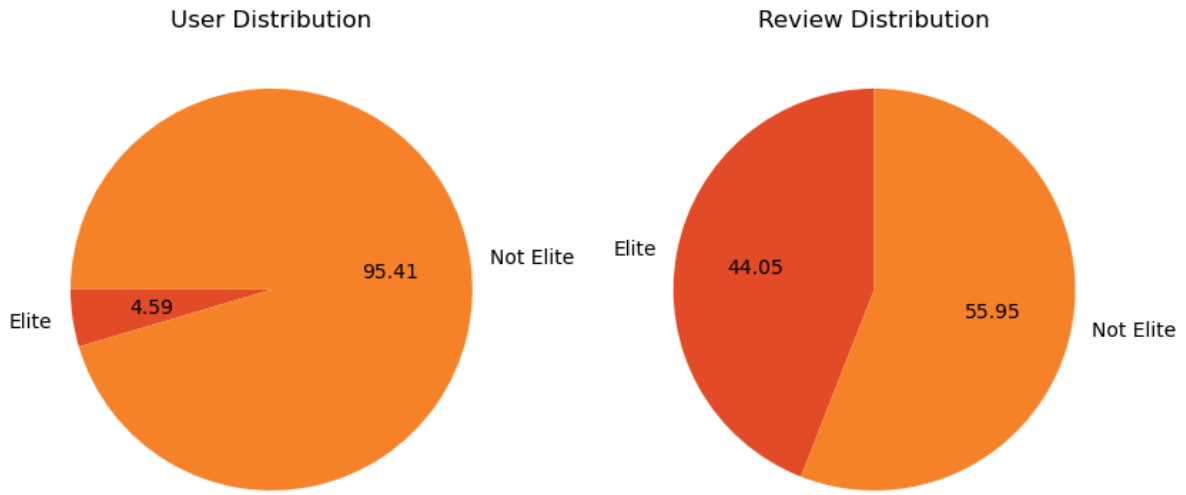
In [41]:
```
# Is there any difference in engagement of elite users and non elite users?
elite_df = pd.read_sql_query("""SELECT
elite,
COUNT(*) AS num_users,
SUM(review_count) AS total_review_count
FROM
(SELECT CASE WHEN elite = '' THEN 'Not Elite' ELSE 'Elite' END AS elite,u.review_cc
FROM user u) AS user_elite
GROUP BY elite""",conn)
```

In [42]:
```
elite_df.head()
```

Out[42]:

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

In [43]:
```
plt.figure(figsize = (10,6))
plt.subplot(1,2,1)
plt.title('User Distribution')
plt.pie(elite_df['num_users'],labels = elite_df['elite'],autopct = '%.2f',startangl

plt.subplot(1,2,2)
plt.title('Review Distribution')
plt.pie(elite_df['total_review_count'],labels = elite_df['elite'],autopct = '%.2f',
plt.show()
```



In [44]:
```
# What are the busiest hours for restaurants?
```

In [45]:
```
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)
```

In [46]:
```
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)
```

In [50]:
```
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").str

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

In [48]:
```

```

Out[48]:

	0
0	

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```