In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
```

In [2]:
```
#Import data set from files
employment=pd.read_csv('Employment - City - Weekly.csv')
geoIDs=pd.read_csv('GeoIDs - City.csv')
mobility=pd.read_csv('Google Mobility - City - Daily.csv')
covid=pd.read_csv('COVID - City - Daily.csv')
consumer_spending=pd.read_csv('Affinity - City - Daily.csv')
econ_small_bussiness=pd.read_csv('Womply - City - Weekly.csv')
```

In [3]:

```
#reformat the employment dataset
employment=employment.rename(columns={"day_endofweek": "day"})
employment = employment.drop(employment[employment.cityid == 45].index)
employment['emp']=[float(emp) for emp in employment['emp']]
employment['date'] = pd.to_datetime(employment[['year', 'month', 'day']])
employment=employment.drop(columns=['year','month', 'day'])
employment
```

Out[3]:

	cityid	emp	emp_incq1	emp_incq2	emp_incq3	emp_incq4	emp_incmiddle	emp_
0	1	0.00228	.000506	.00154	.00356	.00251	.00238	
1	2	0.00120	-.0033	-.00183	.00556	.000793	.00189	
2	3	0.00211	.00321	.00155	.00344	.000749	.00242	
3	4	0.00133	-.000504	-.0058	.0108	.000674	.00245	
4	5	-0.00202	.000567	-.00826	.00483	-.00328	-.0025	
...	...	...	...	...	...	...	...	
9906	49	-0.01490	-.131	-.0341	.0732	.	.00504	
9907	50	-0.10500	-.36	-.0351	.0081	.	-.0169	
9908	51	-0.16000	-.133	-.156	-.226	.	-.188	
9909	52	-0.21900	-.19	-.317	-.16	.	-.245	
9910	53	-0.09210	-.112	-.102	-.0712	.	-.0875	

9724 rows × 10 columns

In [4]:

```
#Combine employment dataset and the geoId on the cityId
employment_geoId=employment.merge(geoIDs, left_on=['cityid'], right_on = ['cityid'
employment_geoId.sort_values(by=['city_pop2019'], ascending=False)
employment_geoId
```

Out[4]:

	cityid	emp	emp_incq1	emp_incq2	emp_incq3	emp_incq4	emp_incmiddle	emp
0	1	0.002280	.000506	.00154	.00356	.00251	.00238	
1	1	0.001550	-.00992	.00411	.00233	-.000303	.00337	
2	1	0.000917	-.016	.00506	.00166	-.00196	.00365	
3	1	0.001680	-.0283	.00693	.00376	-.000899	.00562	
4	1	0.002630	-.0373	.00559	.00767	.00312	.00645	
...	...	...	...	...	...	...	...	
9719	53	-0.085700	-.114	-.0986	-.0934	.	-.0961	
9720	53	-0.081900	-.104	-.0928	-.0798	.	-.0865	
9721	53	-0.080900	-.0965	-.0931	-.0708	.	-.0824	
9722	53	-0.087000	-.104	-.0969	-.0739	.	-.0859	
9723	53	-0.092100	-.112	-.102	-.0712	.	-.0875	

9724 rows × 17 columns

In [5]:
```
#Select row from dataset with popultion over 5 million
employment_geoId_pol_lar=employment_geoId.loc[employment_geoId.city_pop2019>500000
employment_geoId_pol_lar
```

Out[5]:

	cityid	emp	emp_incq1	emp_incq2	emp_incq3	emp_incq4	emp_incmiddle	emp_
0	1	0.002280	.000506	.00154	.00356	.00251	.00238	
1	1	0.001550	-.00992	.00411	.00233	-.000303	.00337	
2	1	0.000917	-.016	.00506	.00166	-.00196	.00365	
3	1	0.001680	-.0283	.00693	.00376	-.000899	.00562	
4	1	0.002630	-.0373	.00559	.00767	.00312	.00645	
...	...	...	...	...	...	...	...	
556	3	-0.186000	-.401	-.106	-.165	.	-.133	
557	3	-0.183000	-.396	-.102	-.155	.	-.126	
558	3	-0.178000	-.393	-.0861	-.152	.	-.116	
559	3	-0.179000	-.401	-.0838	-.15	.	-.114	
560	3	-0.183000	-.406	-.0928	-.151	.	-.12	

561 rows × 17 columns

In [6]:

```
#Calculate the mean value by date for the big city(population > 5m)
large_city_employment_rate_change=employment_geoId_pol_lar.groupby(['date'])['emp'
large_city_employment_rate_change
```

Out[6]:

	date	emp
0	2020-01-17	0.001863
1	2020-01-24	0.000073
2	2020-01-31	-0.000636
3	2020-02-07	0.000183
4	2020-02-14	0.000463
...	...	...
182	2023-07-14	-0.181667
183	2023-07-21	-0.181667
184	2023-07-28	-0.181667
185	2023-08-04	-0.187667
186	2023-08-11	-0.195333

187 rows × 2 columns

In [7]:

```
#Select row from dataset with popultion less than half million
employment_geoId_pol_sml=employment_geoId.loc[employment_geoId.city_pop2019<500000
employment_geoId_pol_sml
```

Out[7]:

	cityid	emp	emp_incq1	emp_incq2	emp_incq3	emp_incq4	emp_incmiddle	emp
8041	44	0.000259	-.00671	.0126	-.000385	-.00433	.00637	
8042	44	0.000482	.0143	-.0166	.00309	-.00396	-.00712	
8043	44	0.000003	.0142	-.0149	-.000864	-.0034	-.00814	
8044	44	-0.001000	.00949	-.0155	-.000983	.000859	-.0085	
8045	44	-0.000186	-.000433	-.0246	.0122	.0203	-.00692	
...	...	...	...	...	...	...	...	
9345	51	-0.134000	-.0986	-.127	-.214	.	-.166	
9346	51	-0.139000	-.0876	-.15	-.211	.	-.178	
9347	51	-0.147000	-.0965	-.158	-.216	.	-.184	
9348	51	-0.152000	-.111	-.157	-.219	.	-.185	
9349	51	-0.160000	-.133	-.156	-.226	.	-.188	

561 rows × 17 columns

In [8]:

```
#Calculate the mean value by date for the big city(population < 0.5m)
small_city_employment_rate_change = employment_geoId_pol_sml.groupby(['date'])['em
small_city_employment_rate_change
#employment_trend_lar['date'] = pd.to_datetime(employment_geoId_pol_lar[['year', '
#employment_trend_lar=employment_trend_lar[['date', 'emp']]
#employment_trend_sm['date'] = [x for x,y,z in employment_trend_sm['year','month',
#employment_trend_lar
```

Out[8]:

	date	emp
0	2020-01-17	-0.003894
1	2020-01-24	0.002386
2	2020-01-31	0.006378
3	2020-02-07	0.010128
4	2020-02-14	0.013128
...	...	...
182	2023-07-14	-0.107600
183	2023-07-21	-0.112633
184	2023-07-28	-0.117933
185	2023-08-04	-0.124093
186	2023-08-11	-0.135967

187 rows × 2 columns

In [9]:
```
#Calculate the mean value by date for all city
employment_rate_change = employment_geoId.groupby(['date'])['emp'].mean().reset_in
employment_rate_change
```

Out[9]:

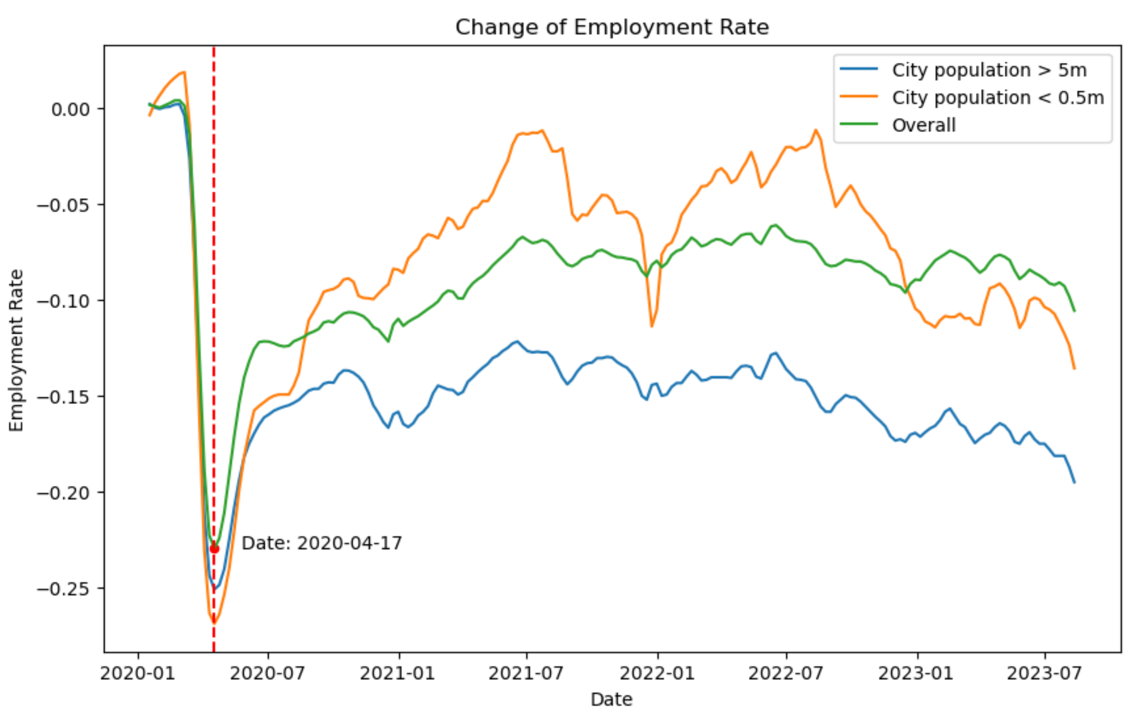
	date	emp
0	2020-01-17	0.001482
1	2020-01-24	0.000732
2	2020-01-31	-0.000031
3	2020-02-07	0.001115
4	2020-02-14	0.002320
...	...	...
182	2023-07-14	-0.092503
183	2023-07-21	-0.091203
184	2023-07-28	-0.093081
185	2023-08-04	-0.098651
186	2023-08-11	-0.105910

187 rows × 2 columns

In [10]:

```
#Draw line plot of the Change of Employment Rate
plt.figure(figsize=(10, 6))
plt.plot(large_city_employment_rate_change['date'], large_city_employment_rate_cha
plt.plot(small_city_employment_rate_change['date'], small_city_employment_rate_cha
plt.plot(employment_rate_change['date'], employment_rate_change['emp'], label='Ove

peak_value = min(employment_rate_change['emp'])
peak_date = employment_rate_change.loc[employment_rate_change['emp'] == peak_value
print(str(peak_date)[:10])

print('min value of large city: ', min(large_city_employment_rate_change['emp']))
print('min value of overall: ', min(employment_rate_change['emp']))
print('min value of small city: ', min(small_city_employment_rate_change['emp']))

plt.axvline(x=peak_date, color='red', linestyle='--')
plt.scatter(peak_date, peak_value, color='red', s=20)
plt.annotate(f'Date: {str(peak_date)[:10]}', (peak_date, peak_value),textcoords="o

plt.title('Change of Employment Rate')
plt.xlabel('Date')
plt.ylabel('Employment Rate')

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

```
2020-04-17
min value of large city:  -0.25133333333333335
min value of overall:  -0.23001923076923078
min value of small city:  -0.26933333333333337
```



In [11]:
```
covid
```

Out[11]:

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_cou
0	2020	1	1	1	.	.	.	
1	2020	1	1	2	.	.	.	
2	2020	1	1	3	.	.	.	
3	2020	1	1	4	.	.	.	
4	2020	1	1	5	.	.	.	
...	...	...	...	...	...	...	...	
64679	2023	6	1	2	2736550	45257	190	
64680	2023	6	2	2	2736745	45258	195	
64681	2023	6	3	2	2736944	45259	199	
64682	2023	6	4	2	2737141	45260	197	
64683	2023	6	5	2	2737341	45261	199	

64684 rows × 28 columns

In [12]:

```
#Clean up the data set of Covid, remove empty row and sort it by population
covid=covid.drop(covid[covid.new_case_count == '.'].index)
covid['date'] = pd.to_datetime(covid[['year', 'month', 'day']])
covid_emp=covid.merge(employment, left_on=['date','cityid'], right_on = ['date','c

covid_emp=covid_emp.merge(geoIDs, left_on=['cityid'], right_on = ['cityid'])
covid_emp.sort_values(by='city_pop2019')
covid_emp
```

Out[12]:

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_coun
0	2020	1	31	1	1	0	0	
1	2020	2	7	1	1	0	0	
2	2020	2	14	1	1	0	0	
3	2020	2	21	1	1	0	0	
4	2020	2	28	1	1	0	0	
...	...	...	...	...	...	...	...	..
8569	2023	3	31	49	161739	1140	238	(
8570	2023	4	7	49	161275	1140	212	(
8571	2023	4	14	49	161063	1140	151	(
8572	2023	4	21	49	160912	1140	139	(
8573	2023	4	28	49	160773	1140	127	(

8574 rows × 44 columns

In [13]:
```
#Select the row by the date value 2020-04-17, located the min. value of employment
top_city=covid_emp.loc[covid_emp['date'] == '2020-04-17']
top_city=top_city.sort_values(by='city_pop2019', ascending=False )
top_city
```

Out[13]:

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_coun
11	2020	4	17	1	10039	370	2961	25
1858	2020	4	17	2	131363	12733	25067	452
181	2020	4	17	3	16369	596	6919	36
2028	2020	4	17	4	3958	53	1199	2
351	2020	4	17	5	2106	56	663	2
1026	2020	4	17	6	1943	55	462	2
5800	2020	4	17	38	7742	139	2524	11
3182	2020	4	17	7	1897	40	653	3
2193	2020	4	17	8	2494	109	594	4
2358	2020	4	17	9	4611	307	785	5
3346	2020	4	17	10	924	28	470	1
1194	2020	4	17	11	831	32	294	1
520	2020	4	17	12	1716	63	386	2
3510	2020	4	17	13	12114	835	2710	43
2853	2020	4	17	41	949	29	318	2
3674	2020	4	17	14	7195	196	3042	18
1527	2020	4	17	35	798	29	244	[illegible]
3018	2020	4	17	43	823	17	248	[illegible]
7111	2020	4	17	15	1077	17	470	[illegible]
7274	2020	4	17	16	882	14	432	1
5964	2020	4	17	42	618	42	391	3
6292	2020	4	17	46	1242	33	331	2
1693	2020	4	17	50	1237	8	445	[illegible]
2688	2020	4	17	39	500	1	175	[illegible]
3838	2020	4	17	17	1012	16	230	1
2523	2020	4	17	36	1753	59	620	2
5472	2020	4	17	33	692	34	276	3
5636	2020	4	17	34	245	6	120	[illegible]
6456	2020	4	17	48	361	6	37	[illegible]
4002	2020	4	17	18	3124	161	925	8
4166	2020	4	17	19	722	14	199	[illegible]
7926	2020	4	17	31	1819	101	438	4
4330	2020	4	17	20	1382	29	599	1
8252	2020	4	17	47	486	3	190	[illegible]

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_coun
689	2020	4	17	21	969	16	261	7
7437	2020	4	17	22	344	4	230	5
4494	2020	4	17	24	411	22	150	11
858	2020	4	17	25	6015	128	2738	117
7763	2020	4	17	26	470	20	107	5
4658	2020	4	17	27	555	49	105	21
4822	2020	4	17	28	1404	56	523	24
6784	2020	4	17	52	650	42	158	16
4986	2020	4	17	29	2098	65	816	48
8089	2020	4	17	37	236	9	74	6
5144	2020	4	17	30	1442	17	373	7
5308	2020	4	17	32	466	17	107	9
6948	2020	4	17	53	354	19	76	6
7600	2020	4	17	23	987	29	588	25
1361	2020	4	17	40	267	8	51	1
8415	2020	4	17	49	553	8	77	3
6620	2020	4	17	51	252	3	41	4
6128	2020	4	17	44	5718	270	490	92

52 rows × 44 columns

In [14]:

```
#Choose 3 city representing the big city
top_city = top_city.iloc[0:3]
top_city
```

Out[14]:

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_coun
11	2020	4	17	1	10039	370	2961	254
1858	2020	4	17	2	131363	12733	25067	4529
181	2020	4	17	3	16369	596	6919	362

3 rows × 44 columns

In [15]:
```
#Calculate the infection rate
ratio_top=[{'city':row['cityname'],'infection rate':float(row['case_count'])/float
ratio_top
```

Out[15]:
```
[{'city': 'Los Angeles',
  'infection rate': 0.0009999893416814862,
  'emp rate': -0.232,
  'new case count': '2961'},
 {'city': 'New York City',
  'infection rate': 0.015756972955025882,
  'emp rate': -0.327,
  'new case count': '25067'},
 {'city': 'Chicago',
  'infection rate': 0.003178302806882718,
  'emp rate': -0.195,
  'new case count': '6919'}]
```

In [16]:
```
#Choose 3 city from the bottom representing the small population city
bot_city = top_city.iloc[-3:]
bot_city
```

Out[16]:

	year	month	day	cityid	case_count	death_count	new_case_count	new_death_coun
11	2020	4	17	1	10039	370	2961	25
1858	2020	4	17	2	131363	12733	25067	452
181	2020	4	17	3	16369	596	6919	36

3 rows × 44 columns

In [17]:
```
#Calculate the infection rate
ratio_bot=[{'city':row['cityname'],'infection rate':float(row['case_count'])/float
ratio_bot
```

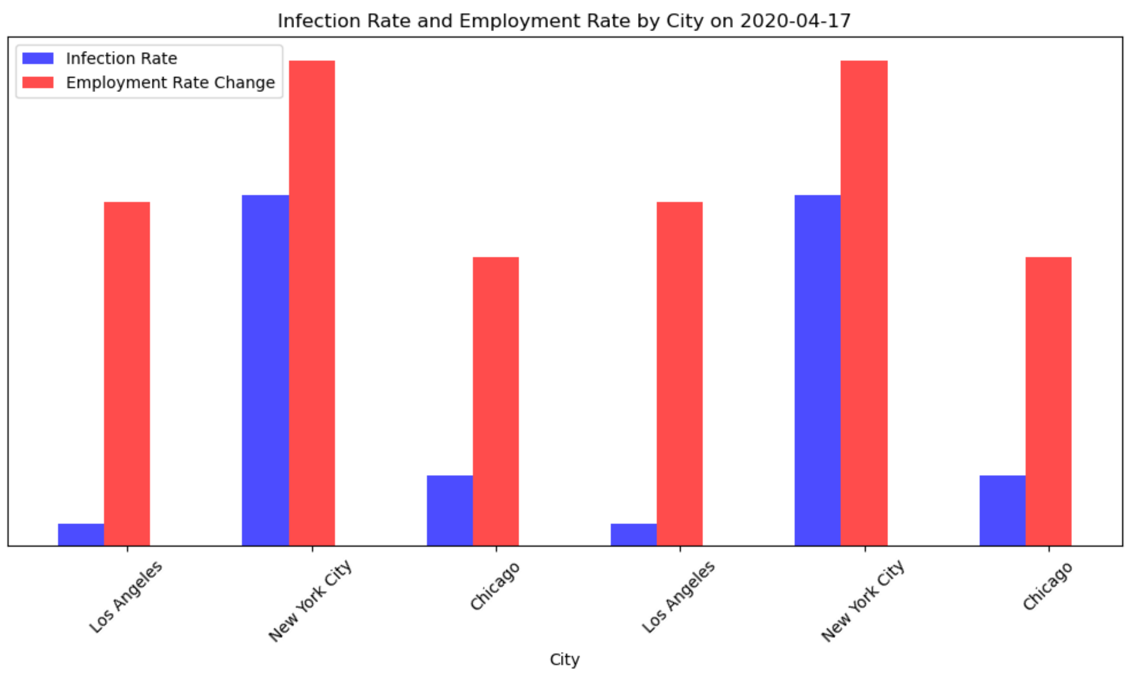
Out[17]:
```
[{'city': 'Los Angeles',
  'infection rate': 0.0009999893416814862,
  'emp rate': -0.232,
  'new case count': '2961'},
 {'city': 'New York City',
  'infection rate': 0.015756972955025882,
  'emp rate': -0.327,
  'new case count': '25067'},
 {'city': 'Chicago',
  'infection rate': 0.003178302806882718,
  'emp rate': -0.195,
  'new case count': '6919'}]
```

In [122]:
```
#Combine the result
ratio=ratio_top+ratio_bot
ratio
```

Out[122]:
```
[{'city': 'Los Angeles',
  'infection rate': 0.0009999893416814862,
  'emp rate': -0.232,
  'new case count': '2961'},
 {'city': 'New York City',
  'infection rate': 0.015756972955025882,
  'emp rate': -0.327,
  'new case count': '25067'},
 {'city': 'Chicago',
  'infection rate': 0.003178302806882718,
  'emp rate': -0.195,
  'new case count': '6919'},
 {'city': 'Los Angeles',
  'infection rate': 0.0009999893416814862,
  'emp rate': -0.232,
  'new case count': '2961'},
 {'city': 'New York City',
  'infection rate': 0.015756972955025882,
  'emp rate': -0.327,
  'new case count': '25067'},
 {'city': 'Chicago',
  'infection rate': 0.003178302806882718,
  'emp rate': -0.195,
  'new case count': '6919'}]
```

In [19]:

```
#Draw a bar chart for the infection rate and employment rate
cities = [item['city'] for item in ratio]
infection_rates = [item['infection rate']*15 for item in ratio]
emp_rates = [abs(item['emp rate']) for item in ratio]
#new_case = [int(item['new case count']) for item in ratio]
#print(new_case)

x = np.arange(len(cities))
width = 0.25

# Create a bar chart for infection rate
plt.figure(figsize=(10, 6))
plt.bar(x - width, infection_rates, width, label='Infection Rate', color='b', alph

# Create a bar chart for employment rate
plt.bar(x , emp_rates, width, label='Employment Rate Change', color='r', alpha=0.7

plt.xlabel('City')
plt.title('Infection Rate and Employment Rate by City on 2020-04-17')
plt.xticks(x, cities, rotation=45)
plt.legend()

plt.gca().axes.get_yaxis().set_visible(False)

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



In [20]:
```
geoIDs
```

Out[20]:

	cityid	cityname	stateabbrev	statename	statefips	lat	lon	city_pop2019
0	1	Los Angeles	CA	California	6	34.05	-118.24	10039107
1	2	New York City	NY	New York	36	40.71	-74.01	8336817
2	3	Chicago	IL	Illinois	17	41.88	-87.63	5150233
3	4	Houston	TX	Texas	48	29.76	-95.37	4713325
4	5	Phoenix	AZ	Arizona	4	33.45	-112.07	4485414
5	6	San Diego	CA	California	6	32.72	-117.16	3338330
6	7	Dallas	TX	Texas	48	32.78	-96.80	2635516
7	8	Las Vegas	NV	Nevada	32	36.17	-115.14	2266715
8	9	Seattle	WA	Washington	53	47.61	-122.33	2252782
9	10	Fort Worth	TX	Texas	48	32.76	-97.33	2102515
10	11	San Antonio	TX	Texas	48	29.42	-98.49	2003554
11	12	San Jose	CA	California	6	37.34	-121.89	1927852
12	13	Detroit	MI	Michigan	26	42.33	-83.05	1749343
13	14	Philadelphia	PA	Pennsylvania	42	39.95	-75.17	1584064
14	15	Columbus	OH	Ohio	39	39.96	-83.00	1316756
15	16	Austin	TX	Texas	48	30.27	-97.74	1273954
16	17	Charlotte	NC	North Carolina	37	35.23	-80.84	1110356
17	18	Indianapolis	IN	Indiana	18	39.77	-86.16	964582
18	19	Jacksonville	FL	Florida	12	30.33	-81.66	957755
19	20	Memphis	TN	Tennessee	47	35.15	-90.05	937166
20	21	San Francisco	CA	California	6	37.77	-122.42	881549
21	22	El Paso	TX	Texas	48	31.78	-106.44	839238
22	23	Baltimore	MD	Maryland	24	39.29	-76.61	593490
23	24	Portland	OR	Oregon	41	45.52	-122.68	812855
24	25	Boston	MA	Massachusetts	25	42.36	-71.06	803907
25	26	Oklahoma City	OK	Oklahoma	40	35.47	-97.52	797434
26	27	Louisville	KY	Kentucky	21	38.25	-85.76	766757
27	28	Denver	CO	Colorado	8	39.74	-104.99	727211
28	29	Washington	DC	District of Columbia	11	38.91	-77.04	705749
29	30	Nashville	TN	Tennessee	47	36.16	-86.78	694144
30	31	Milwaukee	WI	Wisconsin	55	43.04	-87.91	945726
31	32	Albuquerque	NM	New Mexico	35	35.09	-106.61	679121
32	33	Tucson	AZ	Arizona	4	32.22	-110.93	1047279

	cityid	cityname	stateabbrev	statename	statefips	lat	lon	city_pop2019
33	34	Fresno	CA	California	6	36.75	-119.77	999101
34	35	Sacramento	CA	California	6	38.58	-121.49	1552058
35	36	Atlanta	GA	Georgia	13	33.75	-84.39	1063937
36	37	Kansas City	MO	Missouri	29	39.10	-94.58	703011
37	38	Miami	FL	Florida	12	25.76	-80.19	2716940
38	39	Raleigh	NC	North Carolina	37	35.78	-78.64	1111761
39	40	Omaha	NE	Nebraska	31	41.25	-96.00	571327
40	41	Oakland	CA	California	6	37.80	-122.27	1671329
41	42	Minneapolis	MN	Minnesota	27	44.98	-93.27	1265843
42	43	Tampa	FL	Florida	12	27.95	-82.46	1471968
43	44	New Orleans	LA	Louisiana	22	29.95	-90.07	390144
44	45	Wichita	KS	Kansas	20	37.69	-97.34	516042
45	46	Cleveland	OH	Ohio	39	41.50	-81.69	1235072
46	47	Bakersfield	CA	California	6	35.37	-119.02	900202
47	48	Honolulu	HI	Hawaii	15	21.31	-157.86	974563
48	49	Boise	ID	Idaho	16	43.62	-116.20	481587
49	50	Salt Lake City	UT	Utah	49	40.76	-111.89	1160437
50	51	Virginia Beach	VA	Virginia	51	36.85	-75.98	449974
51	52	Colorado Springs	CO	Colorado	8	38.83	-104.82	720403
52	53	Tulsa	OK	Oklahoma	40	36.15	-95.99	651552

In [21]:

```
covid['date'] = pd.to_datetime(covid[['year', 'month', 'day']])
covid=covid.drop(columns=['day'])

covid
```

Out[21]:

	year	month	cityid	case_count	death_count	new_case_count	new_death_count	va
954	2020	1	3	1	0	0	.	
1007	2020	1	3	1	0	0	.	
1058	2020	1	1	1	0	0	.	
1060	2020	1	3	1	0	0	.	
1062	2020	1	5	1	0	0	.	
...	...	...	...	...	...	...	...	
64679	2023	6	2	2736550	45257	190	0	
64680	2023	6	2	2736745	45258	195	0	
64681	2023	6	2	2736944	45259	199	0	
64682	2023	6	2	2737141	45260	197	0	
64683	2023	6	2	2737341	45261	199	0	

61292 rows × 28 columns

In [22]:

```
#Covid data set pre-process
covid=covid[['date','cityid','case_count','new_case_count']]
covid=covid.drop(covid[covid.new_case_count == '.'].index)
covid
```

Out[22]:

	date	cityid	case_count	new_case_count
954	2020-01-25	3	1	0
1007	2020-01-26	3	1	0
1058	2020-01-27	1	1	0
1060	2020-01-27	3	1	0
1062	2020-01-27	5	1	0
...	...	...	...	...
64679	2023-06-01	2	2736550	190
64680	2023-06-02	2	2736745	195
64681	2023-06-03	2	2736944	199
64682	2023-06-04	2	2737141	197
64683	2023-06-05	2	2737341	199

61292 rows × 4 columns

In [23]:

```
covid=covid.drop(covid[covid.new_case_count == '.'].index)
#with pd.option_context('display.max_rows', None, 'display.max_columns', None):
#    print(covid)
```

In [24]:

```
covid['new_case_count']= [int(x) for x in covid['new_case_count']]
covid_case_per_day=covid.groupby(['date'])['new_case_count'].sum().reset_index()
#with pd.option_context('display.max_rows', None, 'display.max_columns', None):
#    print(covid_case_per_day)
```

In [25]:
```
covid_case_per_day
```

Out[25]:

	date	new_case_count
0	2020-01-25	0
1	2020-01-26	0
2	2020-01-27	0
3	2020-01-28	0
4	2020-01-29	0
...	...	...
1223	2023-06-01	190
1224	2023-06-02	195
1225	2023-06-03	199
1226	2023-06-04	197
1227	2023-06-05	199

1228 rows × 2 columns

In [26]:
```
#Consumer spending data set pre-process
consumer_spending
consumer_spending['date'] = pd.to_datetime(consumer_spending[['year', 'month', 'da
consumer_spending=consumer_spending.drop(columns=['year','month', 'day'])
```

In [27]:
```
consumer_spending=consumer_spending[['date','cityid','spend_all']]
consumer_spending=consumer_spending.drop(consumer_spending[consumer_spending.spend
```

In [28]:

```
consumer_spending['spend_all']=[float(spd) for spd in consumer_spending['spend_al
consumer_spending_by_day=consumer_spending.groupby(['date'])['spend_all'].mean().r
consumer_spending_by_day
```

Out[28]:

	date	spend_all
0	2020-01-13	0.001134
1	2020-01-14	-0.009896
2	2020-01-15	-0.009414
3	2020-01-16	-0.017062
4	2020-01-17	-0.021166
...	...	...
941	2023-09-17	0.155013
942	2023-09-24	0.154160
943	2023-10-01	0.132858
944	2023-10-08	0.194251
945	2023-10-15	0.169939

946 rows × 2 columns

In [29]:

```
#Drop two column from the dataset due to lacking of values
mobility=mobility.drop(columns=['gps_transit_stations','gps_transit_stations'])
mobility
```

Out[29]:

	year	month	day	cityid	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_
0	2020	2	24	1	0.00571	-0.00286	
1	2020	2	24	2	0.02000	-0.02410	
2	2020	2	24	3	0.04000	0.02710	
3	2020	2	24	4	0.02140	-0.00714	
4	2020	2	24	5	0.03290	-0.00143	
...	...	...	...	...	...	...	
51140	2022	10	15	49	-0.10100	-0.02000	
51141	2022	10	15	50	-0.11400	-0.07290	
51142	2022	10	15	51	-0.13000	-0.04860	
51143	2022	10	15	52	-0.09860	-0.07710	
51144	2022	10	15	53	-0.00429	0.00143	

51145 rows × 10 columns

In [30]:
```
#Convert it to another dataframe
mobility['date'] = pd.to_datetime(mobility[['year', 'month', 'day']])
mblt_col=["date","cityid","gps_retail_and_recreation","gps_grocery_and_pharmacy","
mobile = mobility[mblt_col]
mobile = mobile.groupby(['date'])['gps_retail_and_recreation','gps_grocery_and_pha
```

```
C:\Users\claio\AppData\Local\Temp\ipykernel_8072\3018456002.py:4: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be dep
recated, use a list instead.
  mobile = mobile.groupby(['date'])['gps_retail_and_recreation','gps_grocery_and_
pharmacy','gps_workplaces','gps_residential',"gps_away_from_home"].mean().reset_i
ndex()
```

In [31]:
```
mobile
```

Out[31]:

	date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_workplaces	gps_resi
0	2020-02-24	0.026122	0.010062	0.010062	0.(
1	2020-02-25	0.033874	0.014156	0.012498	0.(
2	2020-02-26	0.042136	0.019076	0.013270	-0.(
3	2020-02-27	0.052115	0.025338	0.015764	-0.(
4	2020-02-28	0.060467	0.033913	0.018789	-0.(
...	...	...	...	...	
960	2022-10-11	-0.152111	-0.092559	-0.249057	0.(
961	2022-10-12	-0.155723	-0.096561	-0.248472	0.(
962	2022-10-13	-0.158542	-0.099121	-0.250857	0.(
963	2022-10-14	-0.158493	-0.098971	-0.251472	0.(
964	2022-10-15	-0.158334	-0.098445	-0.250975	0.(

965 rows × 6 columns

In [32]:
```
consumer_spending_by_day
```

Out[32]:

	date	spend_all
0	2020-01-13	0.001134
1	2020-01-14	-0.009896
2	2020-01-15	-0.009414
3	2020-01-16	-0.017062
4	2020-01-17	-0.021166
...	...	...
941	2023-09-17	0.155013
942	2023-09-24	0.154160
943	2023-10-01	0.132858
944	2023-10-08	0.194251
945	2023-10-15	0.169939

946 rows × 2 columns

In [33]:

```
#Use linear method to interpolate data between value on date column, which use to
#The result of the interpolated data will compare with the result of small size da
date_range = pd.date_range(start=consumer_spending_by_day['date'].iloc[0], end=con

new_df = pd.DataFrame({'date': date_range})

consumer_merged_df = pd.merge(new_df, consumer_spending_by_day, on='date', how='le

consumer_merged_df['spend_all'].interpolate(method='linear', inplace=True)

consumer_merged_df = consumer_merged_df.reset_index(drop=True)

consumer_merged_df
```

Out[33]:

	date	spend_all
0	2020-01-13	0.001134
1	2020-01-14	-0.009896
2	2020-01-15	-0.009414
3	2020-01-16	-0.017062
4	2020-01-17	-0.021166
...	...	...
1367	2023-10-11	0.183831
1368	2023-10-12	0.180358
1369	2023-10-13	0.176885
1370	2023-10-14	0.173412
1371	2023-10-15	0.169939

1372 rows × 2 columns

In [34]:
```
covid_case_per_day
```

Out[34]:

	date	new_case_count
0	2020-01-25	0
1	2020-01-26	0
2	2020-01-27	0
3	2020-01-28	0
4	2020-01-29	0
...	...	...
1223	2023-06-01	190
1224	2023-06-02	195
1225	2023-06-03	199
1226	2023-06-04	197
1227	2023-06-05	199

1228 rows × 2 columns

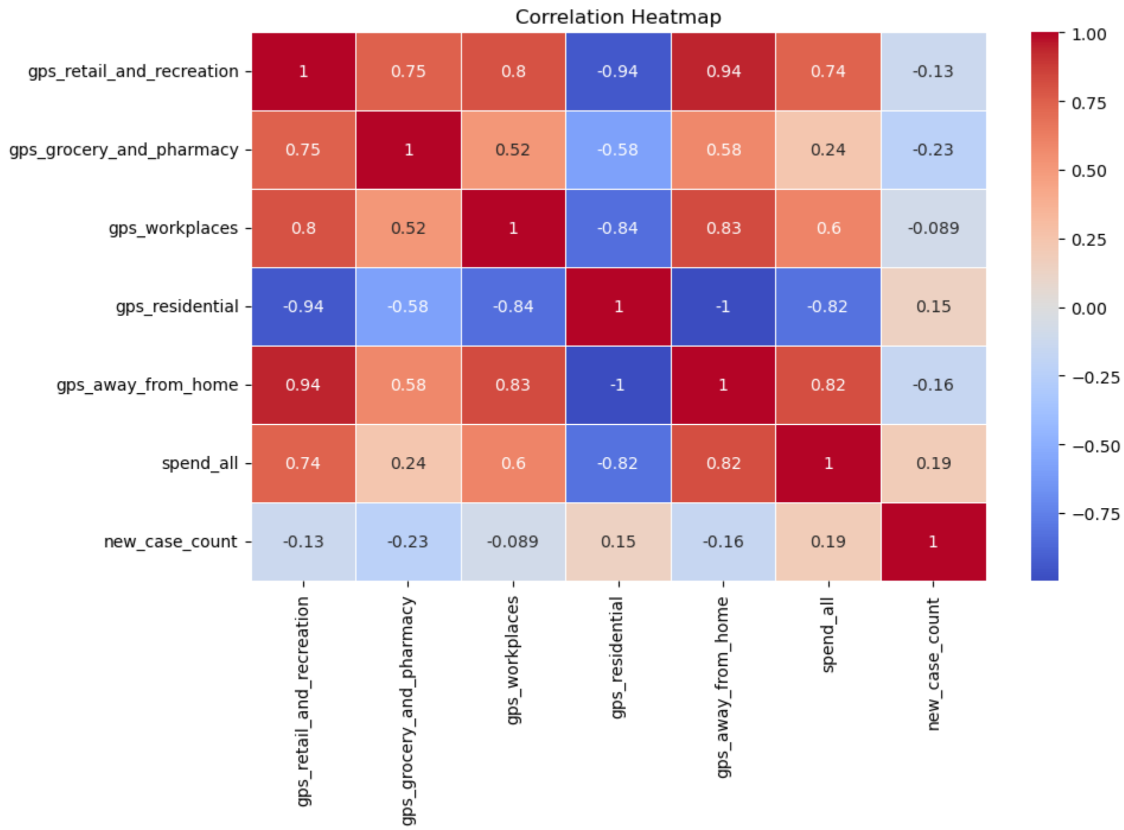
In [35]:
```
#Combine 4 datasets
combine_df = mobile.merge(consumer_merged_df,left_on=['date'], right_on = ['date']
combine_df = combine_df.merge(covid_case_per_day,left_on=['date'], right_on = ['da
combine_df
```

3	2020-02-27	0.052115	0.025338	0.015764
4	2020-02-28	0.060467	0.033913	0.018789
...	...	...	...	...
960	2022-10-11	-0.152111	-0.092559	-0.249057
961	2022-10-12	-0.155723	-0.096561	-0.248472
962	2022-10-13	-0.158542	-0.099121	-0.250857
963	2022-10-14	-0.158493	-0.098971	-0.251472
964	2022-10-15	-0.158334	-0.098445	-0.250975

In [36]:

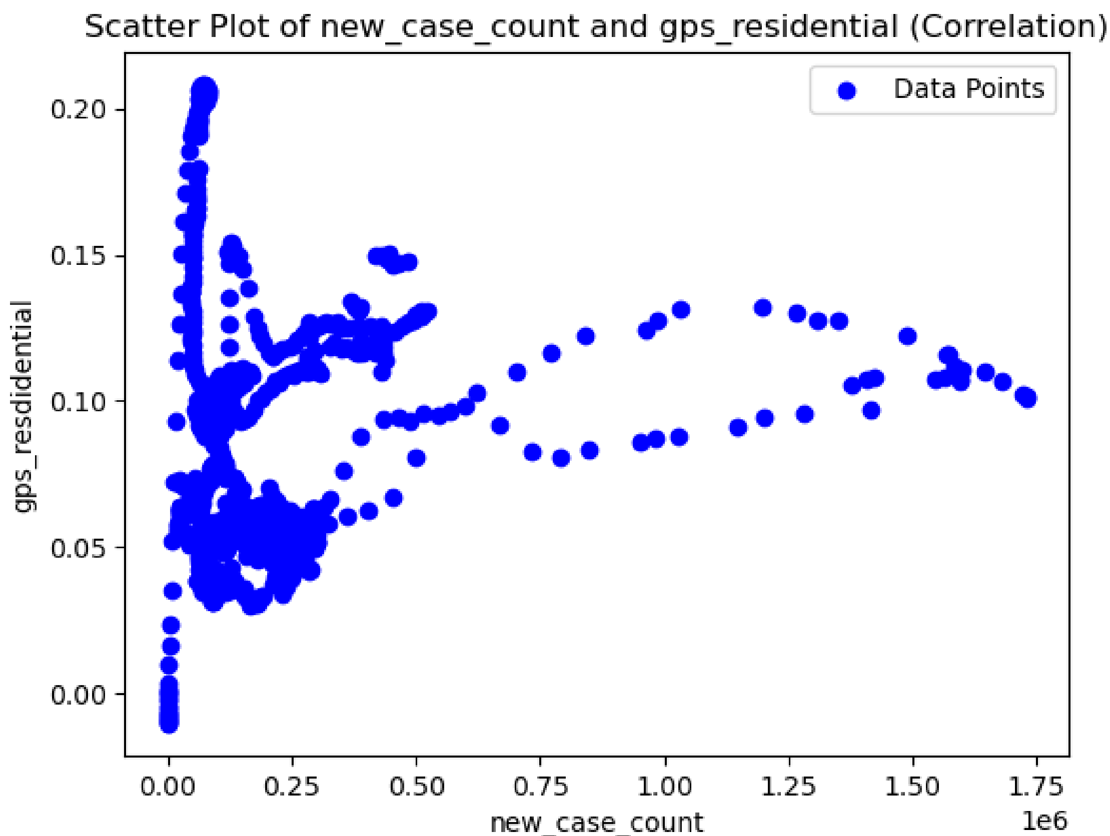
```
#Combine all feature
heatmap_df=combine_df.drop(columns=['date'])
heatmap_df

correlation_matrix = heatmap_df.corr()

# Create a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
```

Out[36]: Text(0.5, 1.0, 'Correlation Heatmap')



In [37]:

```
#Scatter Plot represent the correlation of new covid case and residential info
plt.scatter(combine_df["new_case_count"], combine_df["gps_residential"], c='blue',

plt.xlabel('new_case_count')
plt.ylabel('gps_resdidential')
plt.title('Scatter Plot of new_case_count and gps_residential (Correlation)')

plt.legend()

plt.show()
```



In [38]:

```
#econ_small_bussiness pre-process
econ_small_bussiness['day']=econ_small_bussiness['day_endofweek']
econ_small_bussiness['date'] = pd.to_datetime(econ_small_bussiness[['year', 'month
econ_small_bussiness['date'] = econ_small_bussiness['date'] - pd.DateOffset(days=2
sm_business_col=['date','cityid','merchants_all','revenue_all']
econ_small_bussiness = econ_small_bussiness[sm_business_col]
econ_small_bussiness
sm_b_df=econ_small_bussiness.groupby(['date'])['merchants_all','revenue_all'].mean
sm_b_df
```

```
C:\Users\claio\AppData\Local\Temp\ipykernel_8072\167200411.py:8: FutureWarni
ng: Indexing with multiple keys (implicitly converted to a tuple of keys) wi
ll be deprecated, use a list instead.
  sm_b_df=econ_small_bussiness.groupby(['date'])['merchants_all','revenue_al
l'].mean().reset_index()
```

Out[38]:

	date	merchants_all	revenue_all
0	2020-01-10	0.010253	-0.009718
1	2020-01-17	-0.003797	0.004772
2	2020-01-24	-0.004183	0.011522
3	2020-01-31	-0.002925	-0.003220
4	2020-02-07	-0.001004	0.005327
...	...	...	...
104	2022-01-07	0.042990	0.070065

In [39]:

```
#Date interpolation of small bussiness
date_range = pd.date_range(start=sm_b_df['date'].iloc[0], end=sm_b_df['date'].iloc

new_df = pd.DataFrame({'date': date_range})

merged_df = pd.merge(new_df, sm_b_df, on='date', how='left')

merged_df['merchants_all'].interpolate(method='linear', inplace=True)
merged_df['revenue_all'].interpolate(method='linear', inplace=True)

merged_df = merged_df.reset_index(drop=True)

merged_df
```

Out[39]:

	date	merchants_all	revenue_all
0	2020-01-10	0.010253	-0.009718
1	2020-01-11	0.008246	-0.007648
2	2020-01-12	0.006239	-0.005578
3	2020-01-13	0.004232	-0.003508
4	2020-01-14	0.002225	-0.001438
...	...	...	...
752	2022-01-31	0.019036	0.027497
753	2022-02-01	0.021531	0.041835
754	2022-02-02	0.024026	0.056173
755	2022-02-03	0.026521	0.070511
756	2022-02-04	0.029016	0.084849

757 rows × 3 columns

In [84]:

```
#Employment rate data set combine with city info and count the mean value by date
emp_geo_df=employment_geoId
emp_geo_df=emp_geo_df.groupby(['date'])['emp'].mean().reset_index()
```

In [88]:

```
#Combine ALL data set at weekly basis
combine_emp_rev = combine_df.merge(sm_b_df,left_on=['date'], right_on = ['date'])
combine_emp_rev = combine_emp_rev.merge(emp_geo_df,left_on=['date'], right_on = ['
combine_emp_rev
```

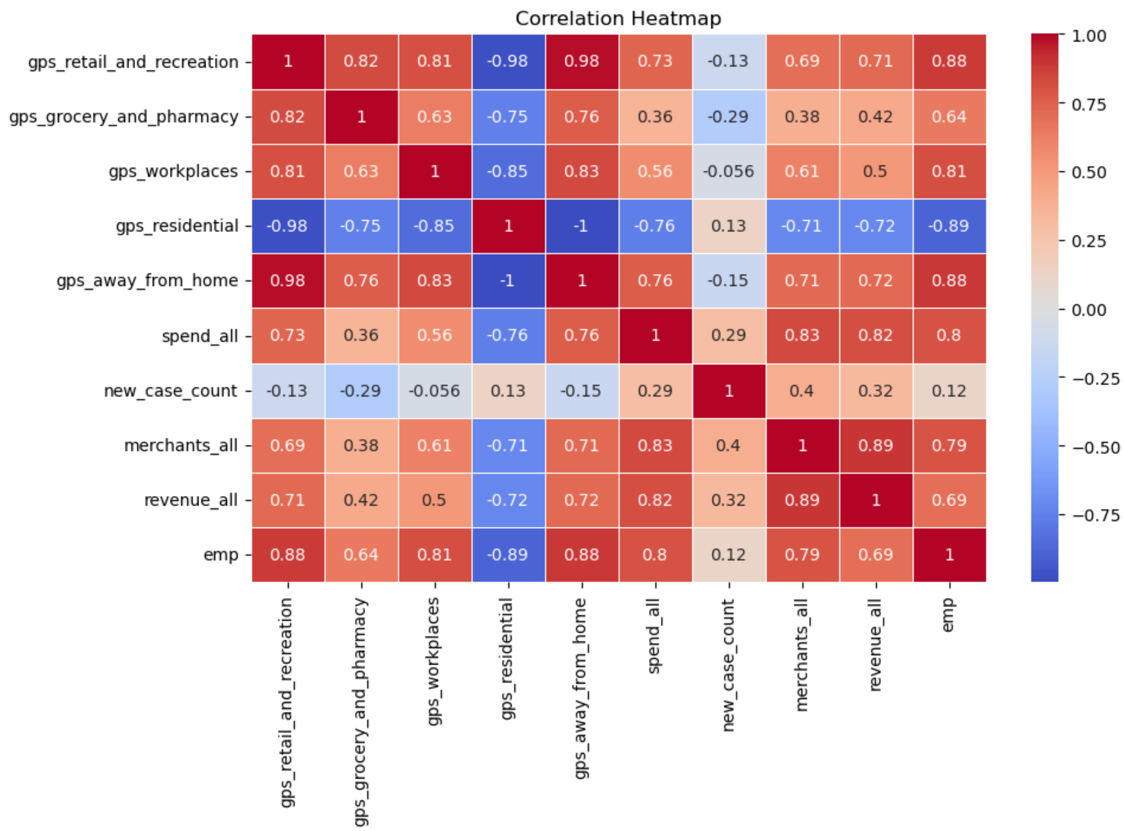
Out[88]:

	date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_workplaces	gps_resi
0	2020-02-28	0.060467	0.033913	0.018789	-0.(
1	2020-03-06	0.079988	0.075869	0.019321	-0.(
2	2020-03-13	0.045159	0.123200	-0.027745	0.(
3	2020-03-20	-0.225755	0.101004	-0.262868	0.'
4	2020-03-27	-0.444208	-0.129394	-0.442283	0.'
...	...	...	...	...	
97	2022-01-07	-0.267385	-0.133570	-0.357472	0.'
98	2022-01-14	-0.228806	-0.094809	-0.281623	0.'
99	2022-01-21	-0.242925	-0.128155	-0.312094	0.'
100	2022-01-28	-0.225234	-0.121674	-0.254472	0.(
101	2022-02-04	-0.235151	-0.120145	-0.281830	0.(

102 rows × 11 columns

In [89]:
```
#Heatmap includes all the factor of analysis
heatmap_df_all=combine_emp_rev.drop(columns=['date'])
heatmap_df_all

correlation_matrix = heatmap_df_all.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
```

Out[89]: Text(0.5, 1.0, 'Correlation Heatmap')



In [90]:

```
#Dataframe of employment and workplace spent time
emp_workplace=combine_emp_rev[["date","emp","gps_workplaces"]]
emp_workplace
```

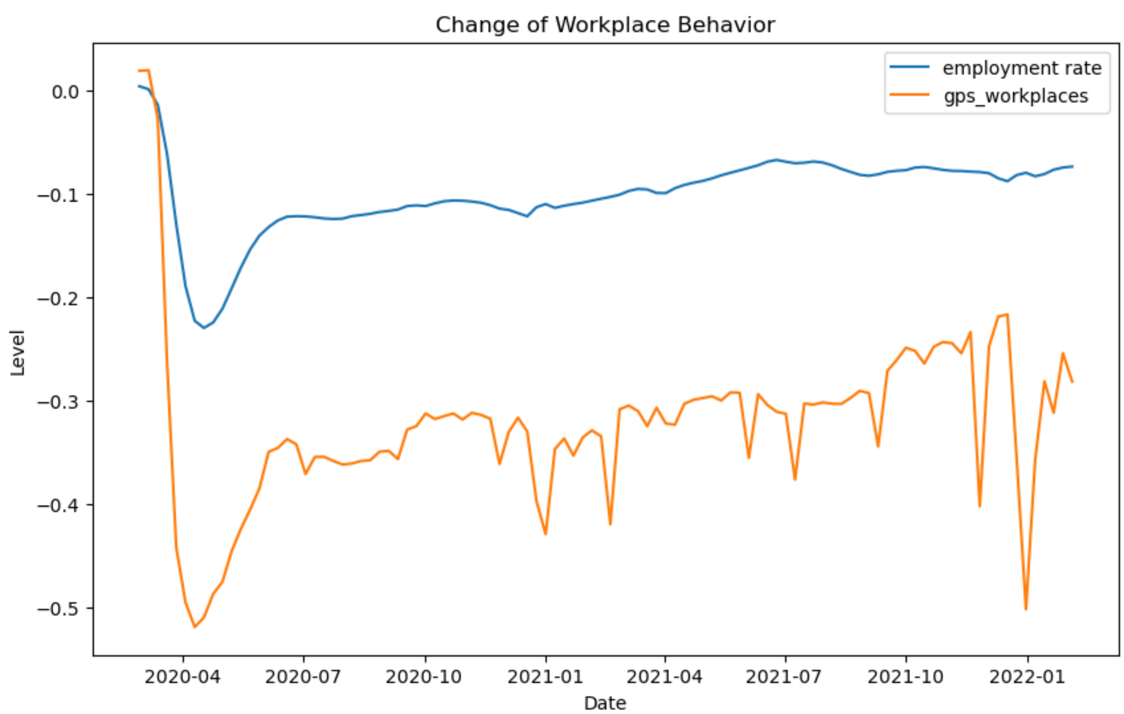
Out[90]:

	date	emp	gps_workplaces
0	2020-02-28	0.003801	0.018789
1	2020-03-06	0.000961	0.019321
2	2020-03-13	-0.014212	-0.027745
3	2020-03-20	-0.061443	-0.262868
4	2020-03-27	-0.129640	-0.442283
...	...	...	...
97	2022-01-07	-0.083267	-0.357472
98	2022-01-14	-0.081131	-0.281623
99	2022-01-21	-0.076905	-0.312094
100	2022-01-28	-0.074794	-0.254472
101	2022-02-04	-0.073885	-0.281830

In [91]:

```
#Reveal the correlation of these two factors in detail
emp_workplace["difference"]=emp_workplace["emp"]-emp_workplace["gps_workplaces"]

plt.figure(figsize=(10, 6))
plt.plot(emp_workplace['date'], emp_workplace['emp'], label='employment rate')
plt.plot(emp_workplace['date'], emp_workplace['gps_workplaces'], label='gps_workpl

plt.title('Change of Workplace Behavior')
plt.xlabel('Date')
plt.ylabel('Level')

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

```
C:\Users\claio\AppData\Local\Temp\ipykernel_8072\4023128233.py:1: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.
org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  emp_workplace["difference"]=emp_workplace["emp"]-emp_workplace["gps_workplace
s"]
```



In [92]:

```
employment_geoId
emp_df=employment_geoId[["emp","date","cityname"]]
emp_df=emp_geo_df.groupby(['date'])['emp'].mean().reset_index()
emp_df
```

Out[92]:

	date	emp
0	2020-01-17	0.001482
1	2020-01-24	0.000732
2	2020-01-31	-0.000031
3	2020-02-07	0.001115
4	2020-02-14	0.002320
...	...	...
182	2023-07-14	-0.092503
183	2023-07-21	-0.091203
184	2023-07-28	-0.093081
185	2023-08-04	-0.098651
186	2023-08-11	-0.105910

187 rows × 2 columns

In [93]:

```
#Data interpolation of employment rate to daily basis
date_range = pd.date_range(start=emp_geo_df['date'].iloc[0], end=emp_geo_df['date'

new_df = pd.DataFrame({'date': date_range})

emp_trend_df = pd.merge(new_df, emp_geo_df, on='date', how='left')

emp_trend_df['emp'].interpolate(method='linear', inplace=True)

emp_trend_df = emp_trend_df.reset_index(drop=True)

emp_trend_df
```

Out[93]:

	date	emp
0	2020-01-17	0.001482
1	2020-01-18	0.001375
2	2020-01-19	0.001268
3	2020-01-20	0.001161
4	2020-01-21	0.001054
...	...	...
1298	2023-08-07	-0.101762
1299	2023-08-08	-0.102799
1300	2023-08-09	-0.103836
1301	2023-08-10	-0.104873
1302	2023-08-11	-0.105910

1303 rows × 2 columns

In [94]:

```
#Combine all the dataset in daily basis
combine_large = combine_df.merge(emp_trend_df,left_on=['date'], right_on = ['date'
combine_large = combine_large.merge(merged_df,left_on=['date'], right_on = ['date'
combine_large
```

Out[94]:

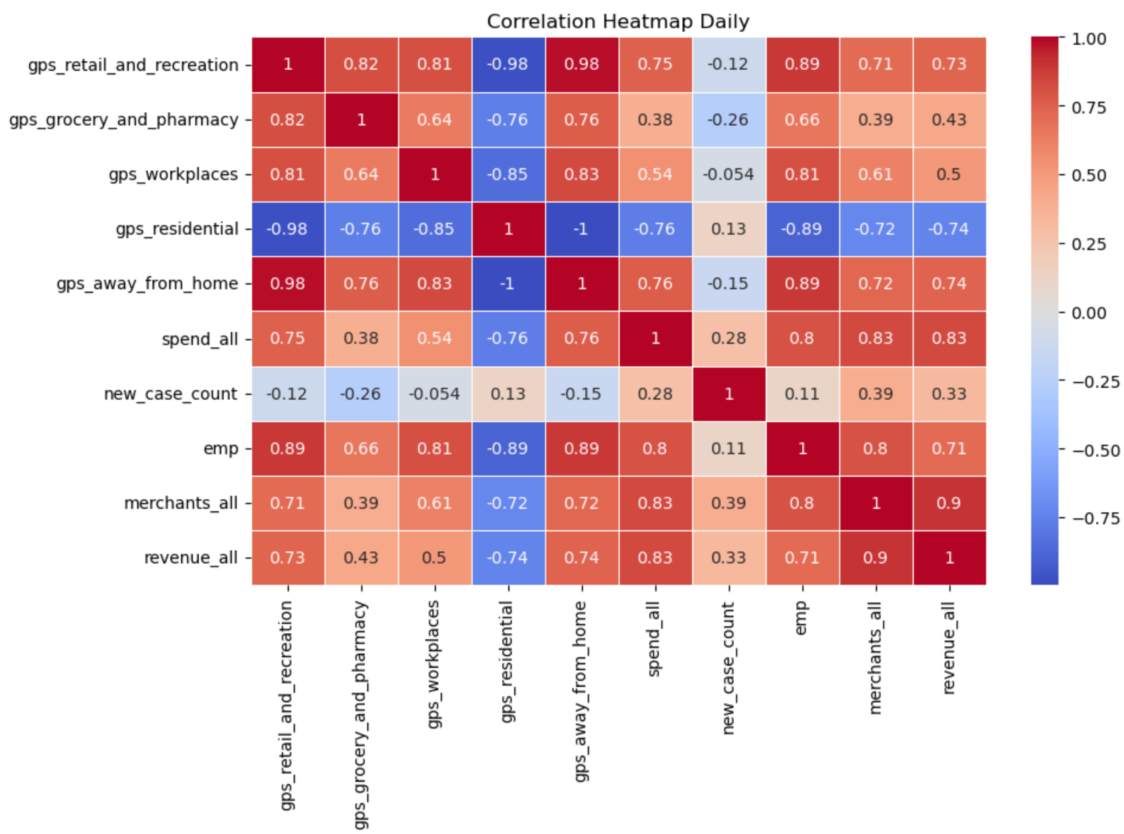
	date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_workplaces	gps
0	2020-02-24	0.026122	0.010062	0.010062	
1	2020-02-25	0.033874	0.014156	0.012498	
2	2020-02-26	0.042136	0.019076	0.013270	
3	2020-02-27	0.052115	0.025338	0.015764	
4	2020-02-28	0.060467	0.033913	0.018789	
...	...	...	...	...	
707	2022-01-31	-0.215987	-0.115928	-0.249868	

In [123]:

```
# Demostrate the daily basis heat map is not much different from the weekly basis
heatmap_df_all_daily=combine_large.drop(columns=['date'])
heatmap_df_all_daily

correlation_matrix = heatmap_df_all_daily.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap Daily')
```

Out[123]:

```
Text(0.5, 1.0, 'Correlation Heatmap Daily')
```



In [95]:

```
#combine the employment rate and spending
emp_spend=combine_large[["emp","spend_all"]]
emp_spend
```

Out[95]:

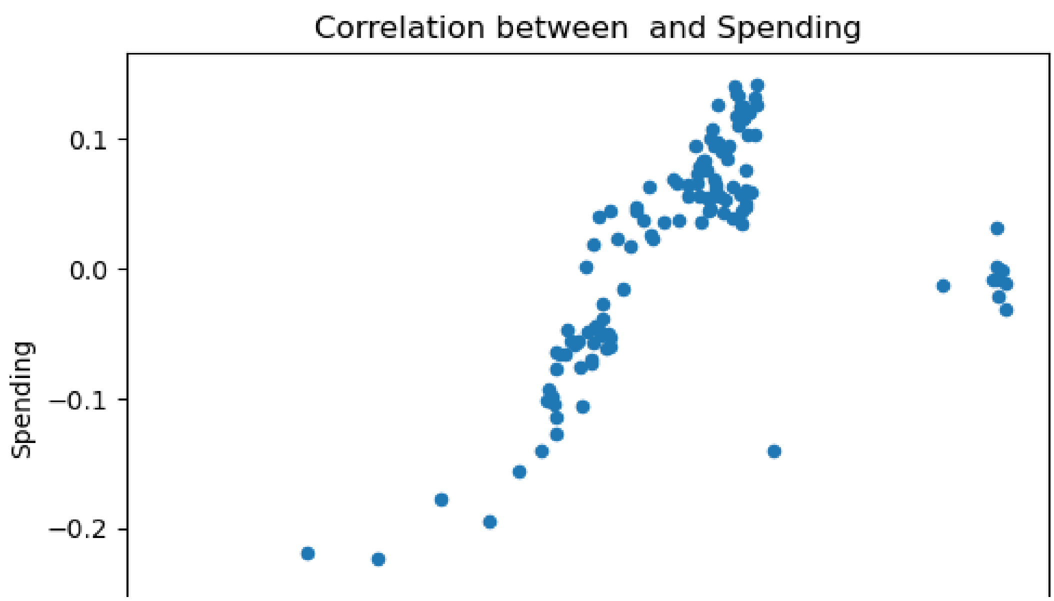
	emp	spend_all
0	0.003754	-0.022519
1	0.003766	-0.025104
2	0.003777	-0.029255
3	0.003789	-0.027846
4	0.003801	-0.031179
...	...	...
707	-0.074404	0.083429
708	-0.074274	0.094303
709	-0.074145	0.098208
710	-0.074015	0.098088
711	-0.073885	0.095169

712 rows × 2 columns

In [96]:

```
#Draw scatter plot to reveal the data distribution
plt.figure(figsize=(10, 6))
emp_spend=consumer_spending_by_day.merge(emp_geo_df,left_on=['date'], right_on = [
emp_spend.plot(kind='scatter', x="emp", y='spend_all')

plt.title('Correlation between  and Spending')
plt.xlabel('Employment Rate')
plt.ylabel('Spending')
plt.show()
```

<Figure size 1000x600 with 0 Axes>



In [97]:

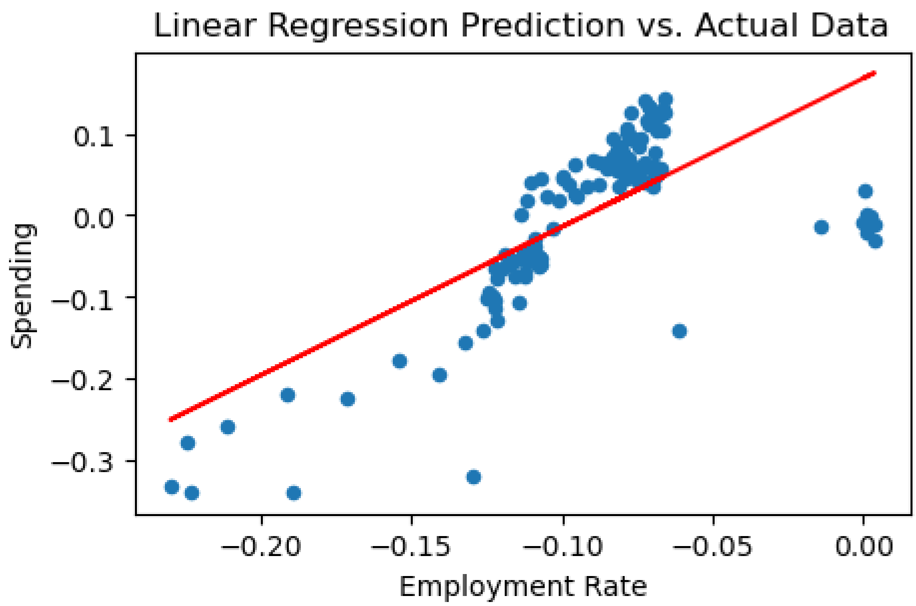
```
#Use linear regression to predict the correlation
import sklearn.linear_model
model = sklearn.linear_model.LinearRegression()
X = np.c_[emp_spend["emp"]]
y = np.c_[emp_spend["spend_all"]]
# Train the model
model.fit(X, y)
```

Out[97]:

```
▾ LinearRegression
LinearRegression()
```

In [98]:

```
#Draw predicted line by intercept and coef value
intercept_value, coeffient = model.intercept_[0], model.coef_[0][0]
emp_spend.plot(kind='scatter', x="emp", y='spend_all', figsize=(5,3))

plt.title('Linear Regression Prediction vs. Actual Data')
plt.plot(X, intercept_value + coeffient*X, "r")
plt.xlabel('Employment Rate')
plt.ylabel('Spending')
plt.show()
```



In [99]:

```
print("Model Intercept", model.intercept_[0])
print("Model Coeffient:", model.coef_[0][0])
```

```
Model Intercept 0.16720764416969355
Model Coeffient: 1.8139710870460024
```

In [100]:

```
#Combine spending and small bussiness dataset to generate a 3 dimension dataset
spend_rev=consumer_spending_by_day.merge(sm_b_df,left_on=['date'], right_on = ['da
spend_rev
```

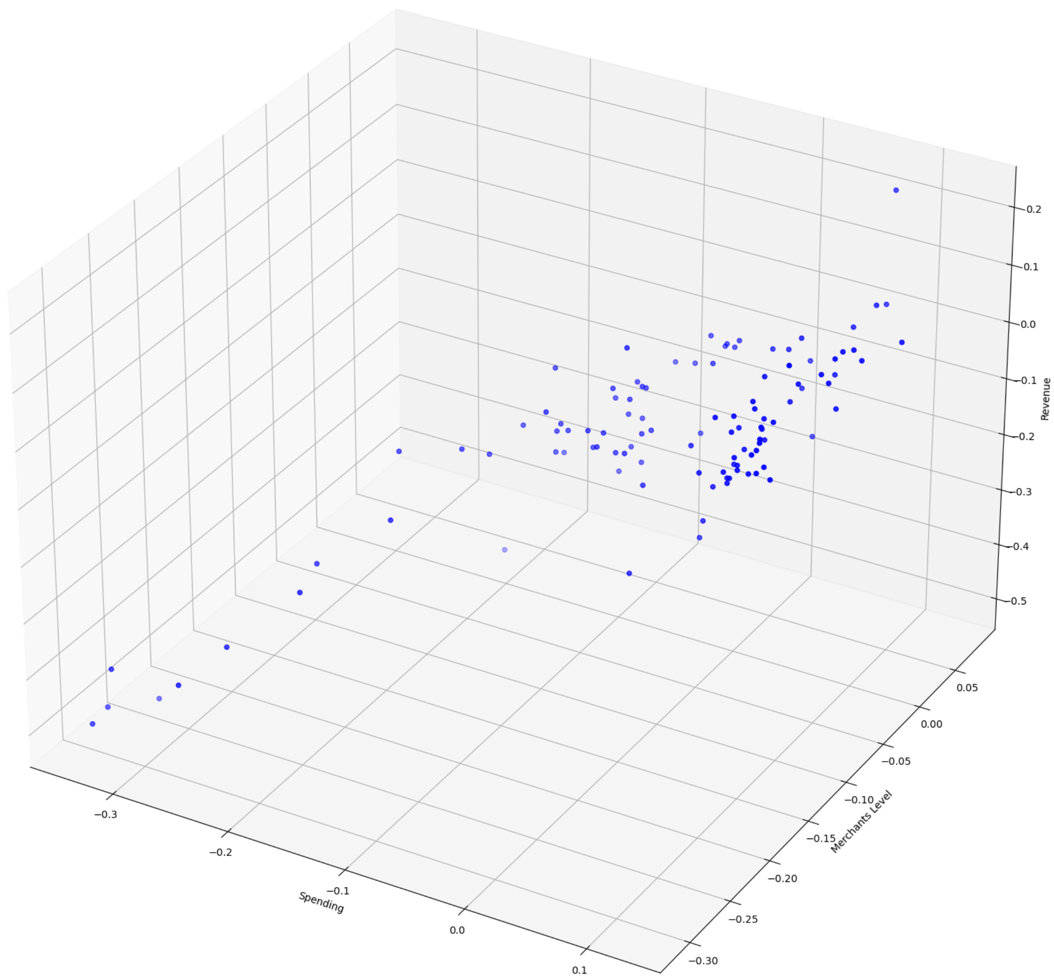
Out[100]:

	date	spend_all	merchants_all	revenue_all
0	2020-01-17	-0.021166	-0.003797	0.004772
1	2020-01-24	0.030892	-0.004183	0.011522
2	2020-01-31	-0.008131	-0.002925	-0.003220
3	2020-02-07	0.000927	-0.001004	0.005327
4	2020-02-14	-0.002106	-0.002155	-0.006601
...	...	...	...	...
103	2022-01-07	0.095070	0.042990	0.070065
104	2022-01-14	0.082719	0.019200	0.051389
105	2022-01-21	0.125645	0.014133	0.055417
106	2022-01-28	0.052608	0.011552	-0.015517
107	2022-02-04	0.095169	0.029016	0.084849

108 rows × 4 columns

In [101]:

```
#Draw the data distribution in 3-d graph
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(spend_rev["spend_all"], spend_rev["merchants_all"], spend_rev["revenue_
ax.set_xlabel('Spending')
ax.set_ylabel('Merchants Level')
ax.set_zlabel('Revenue')

plt.show()
```



In [104]:

```
combine_large
```

Out[104]:

	date	gps_retail_and_recreation	gps_grocery_and_pharmacy	gps_workplaces	gps_resi
0	2020-02-24	0.026122	0.010062	0.010062	0.(
1	2020-02-25	0.033874	0.014156	0.012498	0.(
2	2020-02-26	0.042136	0.019076	0.013270	-0.(
3	2020-02-27	0.052115	0.025338	0.015764	-0.(
4	2020-02-28	0.060467	0.033913	0.018789	-0.(
...	...	...	...	...	
707	2022-01-31	-0.215987	-0.115928	-0.249868	0.(
708	2022-02-01	-0.206025	-0.099810	-0.247394	0.(
709	2022-02-02	-0.210221	-0.096995	-0.251811	0.(
710	2022-02-03	-0.226774	-0.110306	-0.269019	0.(
711	2022-02-04	-0.235151	-0.120145	-0.281830	0.(

712 rows × 11 columns

In [105]:

```
target=combine_large.values[:,-2:]
data=combine_large.values[:,1:-2]
```

In [106]:

```
target
```

Out[106]:

```
array([[-0.003779118598382750, -0.025773396226415096],
       [-0.004929088948787063, -0.030324150943396227],
       [-0.006079059299191376, -0.03487490566037736],
       ...,
       [0.024026091644204856, 0.056173234501347716],
       [0.026520970350404315, 0.0705112398921833],
       [0.029015849056603777, 0.08484924528301888]], dtype=object)
```

In [107]:

```
data
```

Out[107]:

```
array([[0.026121509433962263, 0.010061509433962265, 0.010061698113207548,
        ..., -0.02251877551020408, 11, 0.0037538598901098903],
       [0.03387377358490566, 0.01415566037735849, 0.012498301886792452,
        ..., -0.025103755102040815, 12, 0.0037656016483516483],
       [0.042136415094339616, 0.01907603773083019, 0.013269773584905660,
        ..., -0.029254693877551016, 15, 0.0037773434065934064],
       ...,
       [-0.21022075471698112, -0.0969945283018868, -0.25181132075471696,
        ..., 0.09820795918367348, 732558, -0.07414453296703297],
       [-0.22677358490566038, -0.11030566037735848, -0.2690188679245283,
        ..., 0.09808795918367347, 668328, -0.07401467032967034],
       [-0.2351509433962264, -0.12014528301886793, -0.28183018867924525,
        ..., 0.09516911020408163, 598116, -0.0738848076923077]],
      dtype=object)
```

In [108]:

```
#Pre-processing the multi-factor prediction
from sklearn import model_selection
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(data, target,

# X_train, X_test, Y_train, Y_test
#print(X_train)
print(X_test)
#print(Y_train)
#print(Y_test)
```

```
[[-0.4429056603773585 -0.16324150943396226 -0.4833584905660378 ...
  -0.27716734693877554 60567 -0.21895879120879122]
 [-0.2261452830188679 -0.05779283018867924 -0.3557547169811321 ...
  -0.10085102040816327 151555 -0.12342032967032968]
 [-0.2250698113207547 -0.0795688679245283 -0.3621698113207547 ...
  -0.10140530612244898 151897 -0.12425384615384615]
 ...
 [-0.36713207547169807 -0.08042339622641509 -0.44584905660377355 ...
  -0.21891836734693879 54222 -0.1916134615384615]
 [-0.2540377358490566 -0.1100766037735849 -0.3175471698113207 ...
  -0.041810204081632656 274750 -0.11088818681318681]
 [-0.11478415094339624 -0.03769471698113208 -0.27637735849056605 ...
  0.06025142857142857 190015 -0.07854052197802198]]
```

In [109]:

```
#Using linear regression model to predict
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train)
```

Out[109]:

```
▾ LinearRegression
LinearRegression()
```

In [110]:

```
y_pred = model.predict(X_test)
y_pred
```

```
       [-8.10266985e-02, -1.42863965e-01],
       [-3.09559534e-02, -2.28639031e-02],
       [-7.99439301e-02, -1.37514681e-01],
       [-4.94744728e-02, -4.48728504e-02],
       [-4.10788132e-02, -3.56018580e-02],
       [-7.69859273e-02, -1.03733957e-01],
       [-2.51812207e-02, -1.42235766e-02],
       [-7.93615543e-02, -1.27522690e-01],
       [ 8.59451821e-03, -3.67159756e-02],
       [-1.49272329e-01, -2.48200872e-01],
       [-3.10410642e-02, -9.51843435e-03],
       [-2.66225636e-01, -4.01961943e-01],
       [-4.39246423e-02, -4.21069086e-02],
       [-8.15465058e-02, -1.30780016e-01],
       [-9.42438084e-02, -1.10004754e-01],
       [-3.01180042e-02,  8.63851202e-03],
       [-1.00714948e-01, -1.31527162e-01],
       [-3.95242862e-02, -3.72695364e-03],
       [-1.05205182e-01, -2.28753292e-01],
       [-7.96089452e-02, -1.19912131e-01]
```

In [111]:

```
#Calculate the mean squared error and r squared value
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(Y_test, y_pred)
r2 = r2_score(Y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.001658158620888309
R-squared: 0.8033854245671945
```

In [112]:
```
#Check the moblity dataset
mobility
```

Out[112]:

	year	month	day	cityid	gps_retail_and_recreation	gps_grocery_and_pharmacy
0	2020	2	24	1	0.00571	-0.00286
1	2020	2	24	2	0.02000	-0.02410
2	2020	2	24	3	0.04000	0.02710
3	2020	2	24	4	0.02140	-0.00714
4	2020	2	24	5	0.03290	-0.00143
...	...	...	...	...	...	...
51140	2022	10	15	49	-0.10100	-0.02000

In [113]:
```
#Cleaning dataset by removing empty values, and convert all value to numeric
kmean_mob_df = mobility
kmean_mob_df = kmean_mob_df.drop(kmean_mob_df[kmean_mob_df.gps_parks == '.'].index
kmean_mob_df = kmean_mob_df.drop(kmean_mob_df[kmean_mob_df.gps_residential == '.']
kmean_mob_df = kmean_mob_df.reset_index(drop=True)
```

In [114]:
```
kmean_mob_df["gps_parks"]=[float(x) for x in kmean_mob_df["gps_parks"]]
kmean_mob_df
```

4	2020	2	24	5	0.03290	-0.00143
...	...	...	...	...	...	...
50798	2022	10	15	48	-0.21300	-0.18000
50799	2022	10	15	50	-0.11400	-0.07290
50800	2022	10	15	51	-0.13000	-0.04860
50801	2022	10	15	52	-0.09860	-0.07710
50802	2022	10	15	53	-0.00429	0.00143

50803 rows × 11 columns

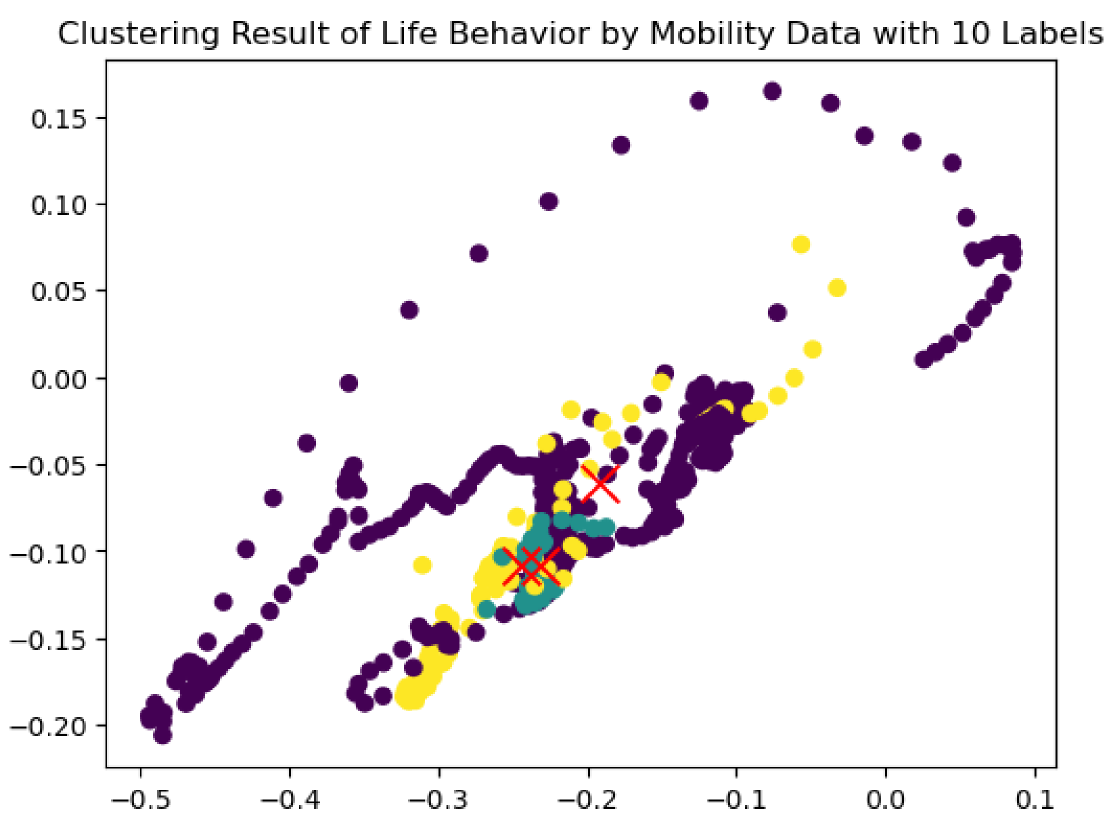
In [118]:
```
#Select data for clustering
mobility_values=kmean_mob_df.values[:,4:-1]
mobility_values
```

Out[118]:
```
array([[0.00571, -0.00286, 0.0714, 0.0214, -0.00143, 0.000625],
       [0.02, -0.0241, 0.139, -0.0326, 0.0116, -0.00907],
       [0.04, 0.0271, 0.281, 0.0257, -0.00857, 0.0147],
       ...,
       [-0.13, -0.0486, 0.33, -0.287, 0.0314, -0.0315],
       [-0.0986, -0.0771, 0.316, -0.199, 0.0214, -0.0202],
       [-0.00429, 0.00143, 0.38, -0.133, 0.02, -0.0215]], dtype=object)
```

In [121]:

```
#Draw a clustering result by using 3 labels
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, completeness_score, homogeneity_sco

kmeans = KMeans(n_clusters=3)

kmeans.fit(data)

labels = kmeans.labels_

silhouette_avg = silhouette_score(data, labels)
print("Silhouette Score:", silhouette_avg)

# Visualize the data and cluster centers (as shown in the previous example)
plt.scatter(data[:, 0], data[:, 1], c=labels)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='
plt.title('Clustering Result of Life Behavior by Mobility Data with 10 Labels')
plt.show()
```

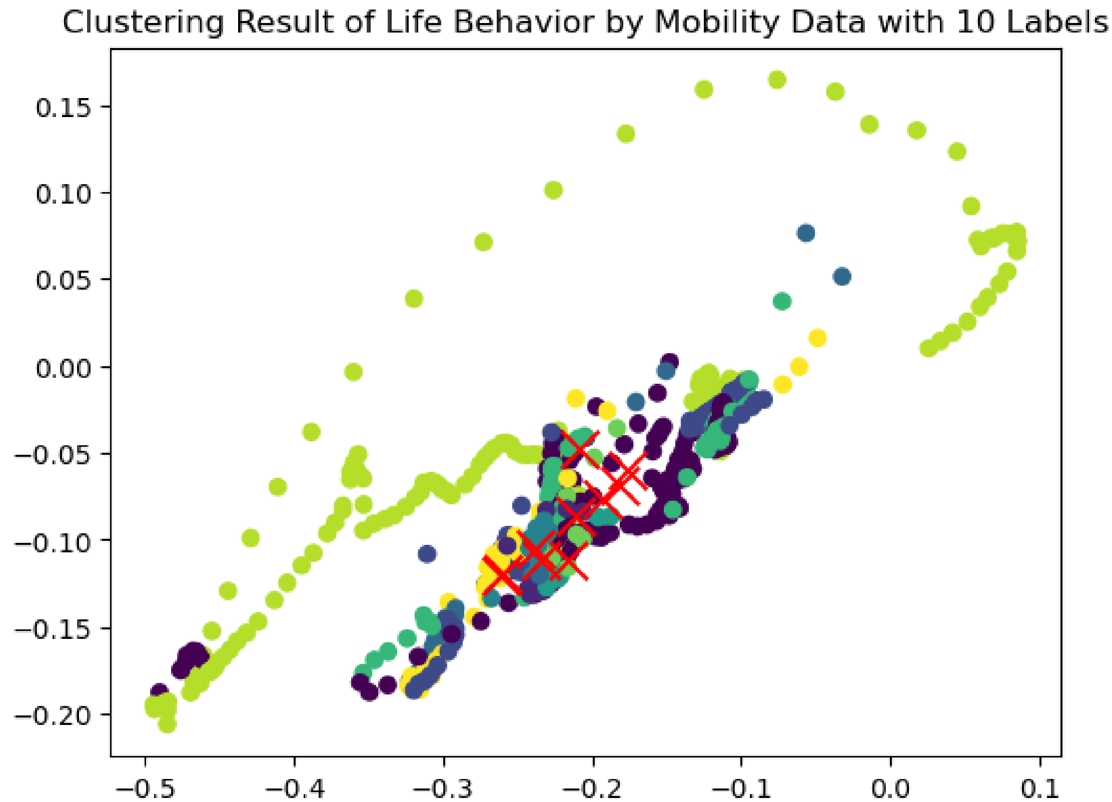
```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(

Silhouette Score: 0.7108765995910622
```



In [119]:

```
#Draw a clustering result by using 10 labels
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, completeness_score, homogeneity_sco

kmeans = KMeans(n_clusters=10)

kmeans.fit(data)

labels = kmeans.labels_

silhouette_avg = silhouette_score(data, labels)
print("Silhouette Score:", silhouette_avg)

# Visualize the data and cluster centers (as shown in the previous example)
plt.scatter(data[:, 0], data[:, 1], c=labels)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='
plt.title('Clustering Result of Life Behavior by Mobility Data with 10 Labels')
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Futur
eWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=3.
  warnings.warn(

Silhouette Score: 0.5486270622481992
```


Clustering Result of Life Behavior by Mobility Data with 10 Labels
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
0.1

In [120]:

```
#Draw a line plot of the result of clustering by using 3-50 labels
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, completeness_score, homogeneity_sco

silhouette_list=[]
number_clusters=[x for x in range(3,40)]

for i in number_clusters:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)

    labels = kmeans.labels_

    silhouette_avg = silhouette_score(data, labels)
    silhouette_list.append(silhouette_avg)

plt.plot(number_clusters, silhouette_list)

plt.title('K-Means Clustering for Different Numbers of Clusters')
plt.xlabel('Numbers of Clusters')
plt.ylabel('Silhouette Score')

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

```
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=3.
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting th
e environment variable OMP_NUM_THREADS=3.
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' i
n 1.4. Set the value of `n_init` explicitly to suppress the warning
```

In [ ]: