In [26]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import matplotlib.cm as cm
import matplotlib.patches as mpatches
import matplotlib.dates as mdates
import datetime as dt
# 设置主题
sns.set(style='white')
# 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams["axes.unicode_minus"]=False #该语句解决图像中的"-"负号的乱码问题
```

In [27]:
```
source_file='./Desktop/dataset/shop_info.txt'
shop_info = pd.read_csv(source_file,
                                names=['shop_id','city_name','location_id','per_pay','score','comment_cnt','shop_level','cate_1_name','cate_2_name','cate_3_name'],
                                dtype={'shop_id': np.int16, 'city_name': 'object', 'location_id': np.int16, 'per_pay': np.int8, 'score': 'object', 'comment_cnt': 'object',
                                       'shop_level': np.int8, 'cate_1_name': 'category', 'cate_2_name': 'category', 'cate_3_name': 'object'},
                                header=None)
```

In [28]:
```
user_pay=pd.read_csv('./Desktop/dataset/user_pay.txt',
                     names=['user_id', 'shop_id', 'time_stamp'],
                     dtype={'user_id': np.int32, 'shop_id': np.int16},
                     header=None)
```

In [29]:
```
user_view=pd.read_csv('./Desktop/dataset/user_view.txt',
                      names=['user_id', 'shop_id', 'time_stamp'],
                      dtype={'user_id': np.int32, 'shop_id': np.int16},
                      header=None)
user_view.columns=['user_id','shop_id','time_stamp']
```

In [30]:
```
shop_info.head()
```

Out[30]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_name
0	1	湖州	885	8	4	12	2	美食
1	2	哈尔滨	64	19	NaN	NaN	1	超市便利店
2	3	南昌	774	5	3	2	0	美食
3	4	天津	380	18	NaN	NaN	1	超市便利店
4	5	杭州	263	2	2	2	0	美食

In [31]:
```
shop_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   shop_id      2000 non-null   int16   
 1   city_name    2000 non-null   object  
 2   location_id  2000 non-null   int16   
 3   per_pay      2000 non-null   int8    
 4   score        1709 non-null   object  
 5   comment_cnt  1709 non-null   object  
 6   shop_level   2000 non-null   int8    
 7   cate_1_name  2000 non-null   category
 8   cate_2_name  2000 non-null   category
 9   cate_3_name  1415 non-null   object  
dtypes: category(2), int16(2), int8(2), object(4)
memory usage: 79.2+ KB
```

In [32]:
```
user_pay.head()
```

Out[32]:

	user_id	shop_id	time_stamp
0	22127870	1862	2015-12-25 17:00:00
1	3434231	1862	2016-10-05 11:00:00
2	16955285	1862	2016-02-10 15:00:00
3	13799128	1862	2016-01-13 14:00:00
4	13799128	1862	2016-07-05 12:00:00

In [33]:
```
user_pay.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69674110 entries, 0 to 69674109
Data columns (total 3 columns):
 #   Column      Dtype 
---  ------      ----- 
 0   user_id     int32 
 1   shop_id     int16 
 2   time_stamp  object
dtypes: int16(1), int32(1), object(1)
memory usage: 930.2+ MB
```

In [34]:
```
user_view.head()
```

Out[34]:

	user_id	shop_id	time_stamp
0	13201967	1197	2016-10-21 18:00:00
1	19461365	1197	2016-06-28 23:00:00
2	15022321	1197	2016-07-16 19:00:00
3	5440872	1197	2016-07-15 07:00:00
4	12594529	1197	2016-08-07 16:00:00

In [35]:
```
user_view.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5556715 entries, 0 to 5556714
Data columns (total 3 columns):
 #   Column      Dtype 
---  ------      ----- 
 0   user_id     int32 
 1   shop_id     int16 
 2   time_stamp  object
dtypes: int16(1), int32(1), object(1)
memory usage: 74.2+ MB
```

检查重复值

In [36]:
```
dup_shop_info= shop_info.duplicated().sum()
print("shop_info发现有 [{}] 条重复数据! ".format(dup_shop_info))
dup_user_pay= user_pay.duplicated().sum()
print("user_pay发现有 [{}] 条重复数据! ".format(dup_user_pay))
dup_user_view= user_view.duplicated().sum()
print("user_view发现有 [{}] 条重复数据! ".format(dup_user_view))
```

```
shop_info发现有 [0] 条重复数据!
user_pay发现有 [2517084] 条重复数据!
user_view发现有 [757364] 条重复数据!
```

重复值删除

In [37]:
```
user_pay.drop_duplicates(keep='first',inplace=True)
user_view.drop_duplicates(keep='first',inplace=True)
```

检查空值

In [38]:
```
print(shop_info.isnull().values.sum())
print(user_pay.isnull().values.sum())
print(user_view.isnull().values.sum())
```

```
1167
0
0
```

In [39]:
```
# 先将 score、comment_cnt 两列通过object类型读入，然后用字符'0'填充NaN，再向下转为最小整形，integer参数让系统自动决定使用最小是哪种整型
shop_info['score'] = pd.to_numeric(shop_info['score'].fillna('0'), downcast='integer')        # 先进行fillna替换NaN的值，然后再改类型，如果反顺序执行，则类型仍为float无法改变
shop_info['comment_cnt'] = pd.to_numeric(shop_info['comment_cnt'].fillna('0'), downcast='integer')
# 处理 cate_3_name 的空值
shop_info['cate_3_name'] = shop_info['cate_3_name'].fillna('无三级分类').astype('category')
```

统计最受欢迎的前 10 类商品(按照二级分类统计)以及人均消费

In [40]:
```
user_pay['time_stamp']=pd.to_datetime(user_pay['time_stamp'])
```

In [41]:
```
#count每个time_stamp商家付款次数
```

In [42]:
```
#as_idex=False在分组后的前面加索引
time_stamp_count=user_pay[['shop_id','time_stamp']].groupby('shop_id',as_index=False).agg('count').rename(columns={'time_stamp':'payment_count'})
```

In [43]:
```
user_pay=user_pay.append(time_stamp_count)
```

In [44]:
```
# 商家付款总次数
```

In [45]:
```
shop_id_payment_count=user_pay.pivot_table(index='shop_id',values='payment_count',aggfunc='sum')
shop_id_payment_count
```

Out[45]:

shop_id	payment_count
1	90242.0
2	37906.0
3	10284.0
4	13398.0
5	31090.0
...	...
1996	6965.0
1997	40084.0
1998	44331.0
1999	93241.0
2000	19640.0

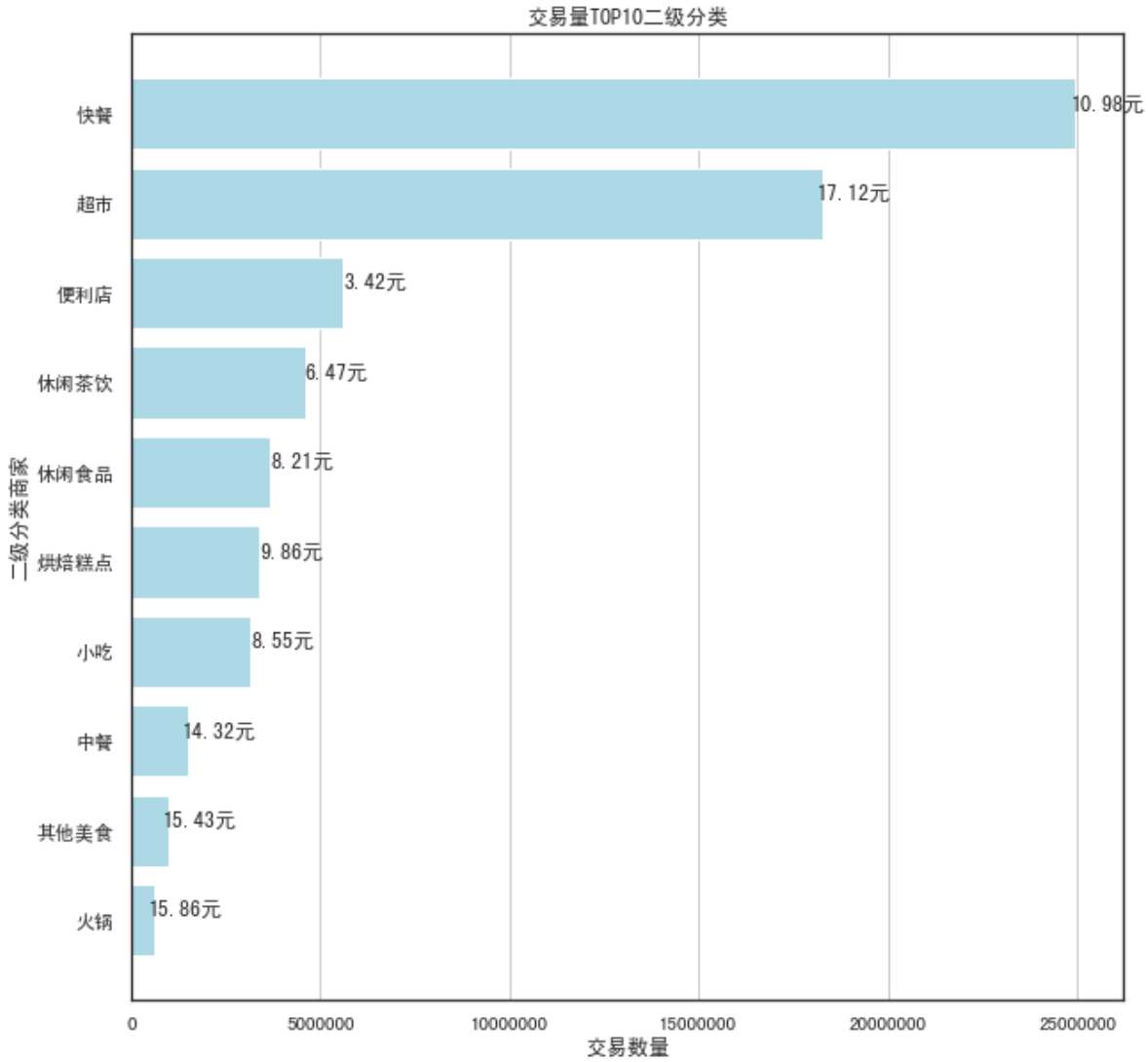
2000 rows × 1 columns

In [46]:
```
date_merge=pd.merge(shop_info,user_pay,left_on='shop_id',right_on='shop_id')
```

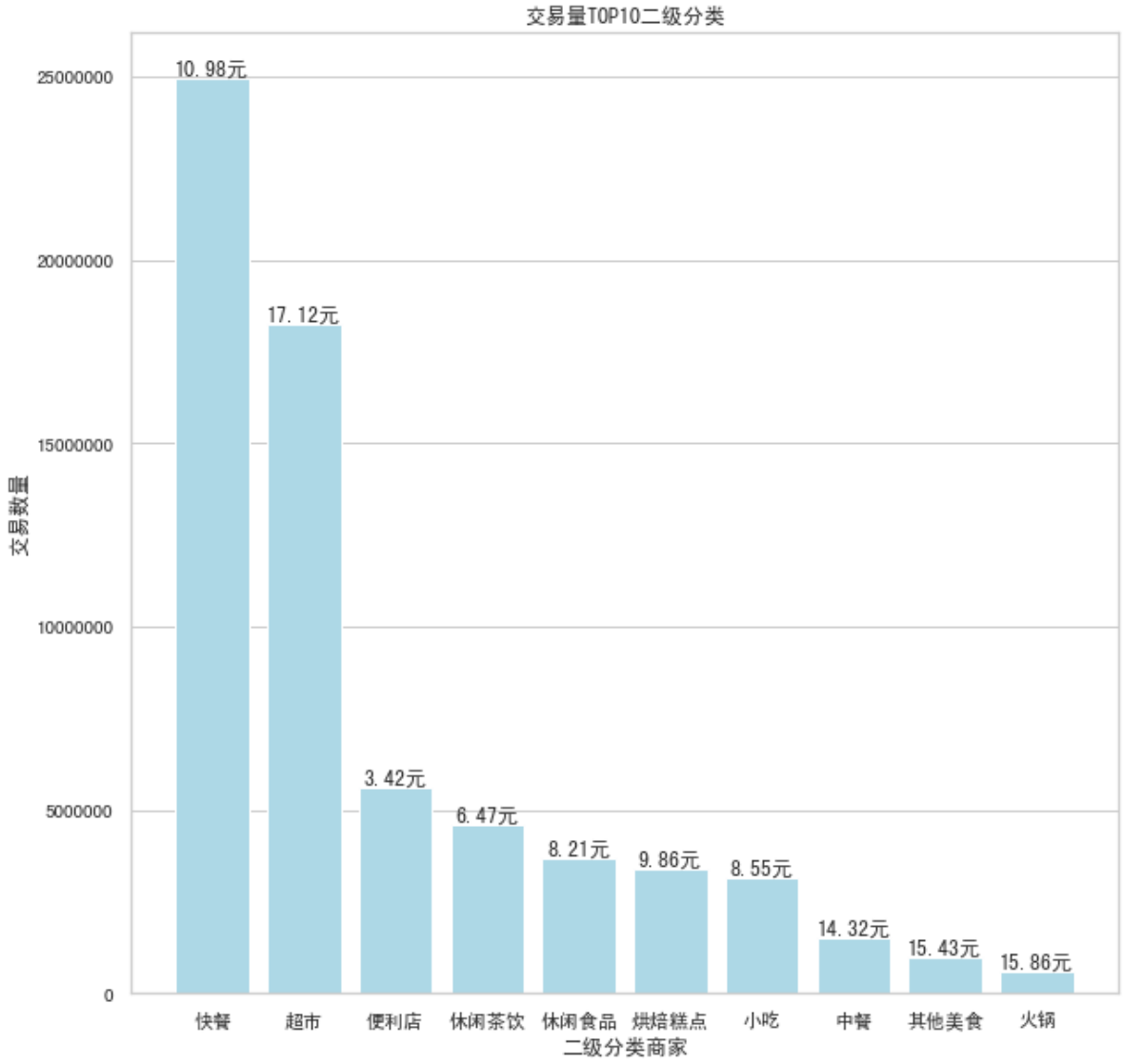
In [47]:
```
result=date_merge[['cate_2_name','payment_count','per_pay']].groupby('cate_2_name',as_index=False).agg({'payment_count':'sum','per_pay':'mean'}).rename(columns={'payment_count':'total_amount'}).sort_values(by='total_amount',ascending=False).head(10)
```

In [48]:
```
#用于tableau可视化文件
result.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau 可视化excel文件/最受欢迎的前 10 类商品(按照二级分类统计)以及人均消费（水平条形图）.xlsx')
```

In [50]:
```
# 二级分类数据划分-第1种画法将y轴翻转，由于上面result是从大到小排列，在水平柱状图中y轴是从0开始从下到上，所以需要把y轴翻转使图形排列正确
fig,ax=plt.subplots(figsize=(10, 10))
y=result['cate_2_name']
width=result['total_amount']
z=result['per_pay']
plt.barh(y,width,color=['lightblue'])
ax.invert_yaxis()#反转y轴使得能从大到小排序，必须要第一行的ax函数才能调用
ax.xaxis.get_major_formatter().set_useOffset(False)# 关闭x轴科学计数法显示
ax.xaxis.get_major_formatter().set_scientific(False)# 关闭x轴科学计数法显示
for a,b,c in zip(y,width,z):
 plt.text(b+c+850000,a,'%.2f元'%c,ha = 'center',va = 'bottom',fontsize=12)#+8500为移动标签的位置
plt.title('交易量TOP10二级分类')
plt.grid(axis='x')#隐藏x轴网格线
plt.xlabel('交易数量')
plt.ylabel('二级分类商家')
plt.show()
```



In [351]:
```
# 二级分类数据划分-第二种画法
fig,ax=plt.subplots(figsize=(10, 10))
x=result['cate_2_name']
y=result['total_amount']
z=result['per_pay']
plt.bar(x,y,color=['lightblue'])
ax.yaxis.get_major_formatter().set_useOffset(False)# 关闭y轴科学计数法显示
ax.yaxis.get_major_formatter().set_scientific(False)# 关闭y轴科学计数法显示
for a,b,c in zip(x,z,y):
 plt.text(a,b+c,'%.2f元'%b,ha = 'center',va = 'bottom',fontsize=12)
plt.title('交易量TOP10二级分类')
plt.xlabel('二级分类商家')
plt.ylabel('交易数量')
plt.grid(axis='x')
plt.show()
# 这边设置的x、y，z值代表了不同柱子在图形中的位置（坐标），通过for循环找到每一个x、z，y值的相应坐标——a,b,c,
# 再使用plt.text在对应位置添文字说明来生成相应的数字标签，而for循环也保证了每一个柱子都有标签。
# 其中，a，b+c表示在每一柱子对应x值、y值上方total_amount的数值+per_pay的数值总和处标注文字说明，'%.2f元' % b，代表标注的文字，%b即每个柱子对应的y值标签，其中2表示保留小数点的后两位，1就表示显示小数后面一位，在f后面是添加文字描述。
# 以此类推； ha='center', va= 'bottom'代表horizontalalignment（水平对齐）、verticalalignment（垂直对齐）的方式，fontsize则是文字大小。条形图、折线图也是如此设置，饼图则在pie命令中有数据标签的对应参数。对于累积柱状图、双轴柱状图则需要用两个for循环，同时通过a与b的不同加减来设置数据标签位置。
```



浏览次数最多的 50 个商家，并输出他们的城市以及人均消费

In [51]:
```
user_view['time_stamp']=pd.to_datetime(user_view['time_stamp'])
```

In [52]:
```
#计算每个商家的用户浏览总次数
shopid_view_count=user_view[['shop_id','time_stamp']].groupby('shop_id',as_index=False).agg('count').rename(columns={'time_stamp':'view_count'})
```

In [53]:
```
shopid_view_count
```

Out[53]:

	shop_id	view_count
0	1	4795
1	2	5520
2	3	534
3	4	4116
4	5	346
...	...	...
1992	1996	572
1993	1997	494
1994	1998	1735
1995	1999	2843
1996	2000	1268

1997 rows × 2 columns

In [54]:
```
data_merge2=pd.merge(shop_info,shopid_view_count,left_on='shop_id',
right_on='shop_id')
```

In [56]:
```
result2=data_merge2[['shop_id','city_name','view_count','per_pay']]
.sort_values(by='view_count',ascending=False).head(50)
result2
```

Out[56]:

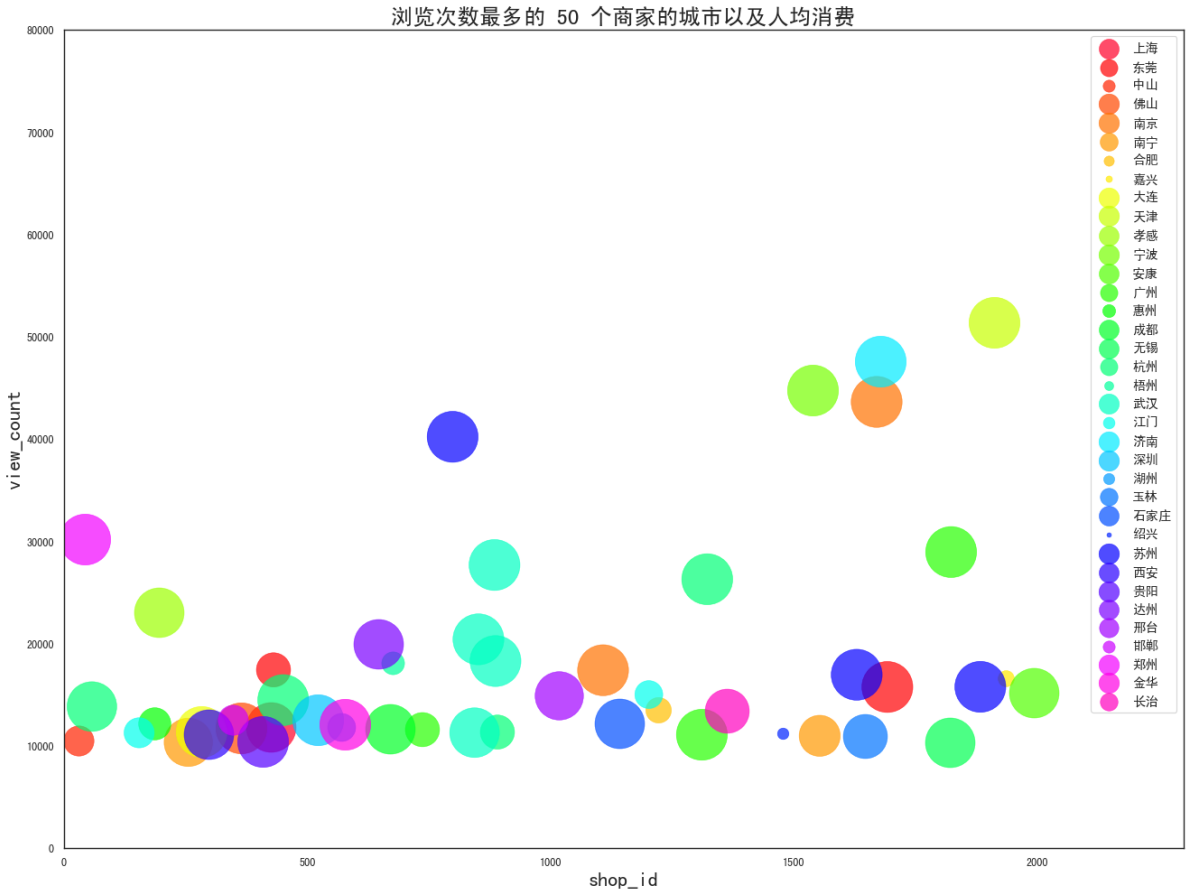
	shop_id	city_name	view_count	per_pay
1907	1911	天津	51409	20
1674	1677	济南	47612	20
1536	1539	宁波	44785	20
1667	1670	南京	43615	20
796	799	苏州	40204	20
43	44	郑州	30239	20
1819	1823	广州	28936	20
882	885	武汉	27670	20
1319	1322	杭州	26279	20
194	195	孝感	23033	19
849	852	武汉	20405	20

644	647	达州	19921	19
884	887	武汉	18279	20
672	675	梧州	18101	4
428	431	东莞	17419	9
1105	1108	南京	17382	20
1626	1629	苏州	16941	20
1931	1935	嘉兴	16599	2
1879	1883	苏州	15760	20
1689	1692	东莞	15746	20
1988	1992	安康	15173	19
1199	1202	江门	15004	6
1014	1017	邢台	14899	18
448	451	杭州	14464	20
57	58	杭州	13830	19
1218	1221	合肥	13506	5
1359	1362	长治	13428	15
345	347	邯郸	12540	7
519	522	深圳	12531	20
1139	1142	石家庄	12215	19
186	187	惠州	12159	8
574	577	金华	12054	20
422	425	上海	11851	19
567	570	湖州	11832	6
362	364	佛山	11707	20
668	671	成都	11616	19
734	737	广州	11574	9
281	283	大连	11429	20
888	891	杭州	11335	9
841	844	武汉	11279	19
154	155	江门	11266	7
1475	1478	绍兴	11191	1

296	298	西安	11143	19
1308	1311	广州	11077	20
1550	1553	南宁	10971	13
1643	1646	玉林	10911	15
29	30	中山	10476	7
405	408	贵阳	10384	20
254	256	南宁	10334	18
1817	1821	无锡	10306	19

In [58]:
```
#用于tableau可视化文件
result2.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau 可视化excel文件/浏览次数最多的 50 个商家的城市以及人均消费（气泡图）.xlsx')
```

In [100]:
```
#预设图像的各种属性
# sns.set(style='white')#设定整体背景主题，第一步载入包即可
# 建立画布
plt.figure(figsize=(20, 15),  # 绘图尺寸默认为（6.4, 4.8）
           dpi = 80,          # 图像分辨效率，默认dpi为100
          facecolor =  "w",   # 背景颜色，默认为白色
          edgecolor = 'k')    # 边框颜色，默认为白色
categories = np.unique(result2["city_name"])   # 使用np.umique对result2["city_name"]去重
# 使用列表推导式，建立colors列表
colors = [plt.cm.gist_rainbow(i/float(len(categories) - 1)) for i in range(len(categories))]
#matplotlib.cm是matplotlib库中内置的色彩映射函数。matplotlib.cm.[色彩]('[数据集]')即对[数据集]应用[色彩]
# 绘图
# 使用函数enumerate：将可遍历的数据对象组合为一个索引序列，同时列出数据和数据索引
for i, category in enumerate(categories):
    plt.scatter('shop_id', 'view_count',
# 横纵坐标
                data=result2.loc[result2.city_name==category, :]
# 横纵坐标所对应的数据
                #, s="view_count"
# 数据尺寸大小
                , s= result2.loc[result2.city_name==category, "per_pay"]*129 #将per_pay设置为气泡大小，由于per_pay数值比较小后面需乘以某个数值调整气泡的尺寸大小，让散点图成为气泡图
                , c= np.array(colors[i]).reshape(1,-1)
                , label=str(category)
# 设定标签名称
                , edgecolors= np.array(colors[i])    # 标记的边缘颜色
                , alpha = 0.7
# 气泡透明度
                , linewidths=.5)
# 线宽
plt.gca().set(xlim = (0,2300),            # 设置x坐标轴的范围
              ylim = (0,80000),           # 设置y坐标轴的范围
             xlabel = "shop_id",           # 设置x坐标的标题
             ylabel = "view_count")        # 设置y坐标的标题
plt.xlabel("shop_id",fontsize=20)           #设置x坐标的标题，字体大小
plt.ylabel("view_count",fontsize=20)        #设置y坐标的标题，字体大小
# plt.xticks(np.arange(0,2500,step=250),fontsize = 20)
# 设置x坐标的字体
# plt.yticks(np.arange(0,80000,step=10000),fontsize = 20)
# 设置y坐标的字体
plt.title("浏览次数最多的 50 个商家的城市以及人均消费", fontsize=22)    # 设置图像标题和字体字体
plt.legend(fontsize = 13,markerscale=0.4)          #makerscales设置图例气泡大小
plt.show()
```



输出北京、上海、广州和深圳四个城市最受欢迎的 5 家奶茶商店和中式快餐编号（最受欢迎是指以下得分最高：0.7 ✖(平均评分/5) + 0.3 ✖ (平均消费金额/最高消费金额)，注：最高消费金额和平均消费金额是从所有消费记录统计出来的）

In [104]:
```
data=shop_info[(shop_info['city_name'].str.contains('北京|上海|广州|深圳'))&(shop_info['cate_3_name'].str.contains('奶茶|中式快餐'))]
data
```

Out[104]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_na
13	14	深圳	862	7	1	4	2	
50	51	北京	659	12	2	3	2	
77	78	上海	699	8	1	1	0	
78	79	上海	1033	15	2	3	1	
92	93	深圳	813	5	0	1	0	
...	...	...	...	...	...	...	...	
1929	1930	上海	993	7	2	2	2	
1930	1931	深圳	1109	4	0	4	0	
1970	1971	上海	642	6	4	4	0	
1973	1974	上海	1127	6	4	3	0	
1996	1997	上海	924	10	1	2	0	

131 rows × 10 columns

In [105]:
```
max_value=data[['city_name','cate_3_name','per_pay']].groupby(['city_name','cate_3_name'],as_index=False).agg(max).round(2)
max_value
```

Out[105]:

	city_name	cate_3_name	per_pay
0	上海	上海本帮菜	NaN
1	上海	东北菜	NaN
2	上海	中式快餐	18.0
3	上海	中式烧烤	NaN
4	上海	其他餐饮美食	NaN
...	...	...	...
171	深圳	面包	NaN
172	深圳	面点	NaN
173	深圳	饮品/甜点	NaN
174	深圳	香锅/烤鱼	NaN
175	深圳	麻辣烫/串串香	NaN

176 rows × 3 columns

In [106]:
```
merge_data_max=pd.merge(data,max_value,on=['city_name','cate_3_name
'],how='left')
merge_data_max
```

Out[106]:

	shop_id	city_name	location_id	per_pay_x	score	comment_cnt	shop_level	cate_1_r
0	14	深圳	862	7	1	4	2	
1	51	北京	659	12	2	3	2	
2	78	上海	699	8	1	1	0	
3	79	上海	1033	15	2	3	1	
4	93	深圳	813	5	0	1	0	
...	...	...	...	...	...	...	...	
126	1930	上海	993	7	2	2	2	
127	1931	深圳	1109	4	0	4	0	
128	1971	上海	642	6	4	4	0	
129	1974	上海	1127	6	4	3	0	
130	1997	上海	924	10	1	2	0	

131 rows × 11 columns

In [107]:
```
merge_data_max.rename(columns={'per_pay_y':'high_pay','per_pay_x':'
per_pay','score_y':'mean_score','score_x':'score'},inplace=True)
merge_data_max
```

Out[107]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_nai
0	14	深圳	862	7	1	4	2	美
1	51	北京	659	12	2	3	2	美
2	78	上海	699	8	1	1	0	美
3	79	上海	1033	15	2	3	1	美
4	93	深圳	813	5	0	1	0	美
...	...	...	...	...	...	...	...	
126	1930	上海	993	7	2	2	2	美
127	1931	深圳	1109	4	0	4	0	美
128	1971	上海	642	6	4	4	0	美
129	1974	上海	1127	6	4	3	0	美
130	1997	上海	924	10	1	2	0	美

131 rows × 11 columns

计算北上广深四个城市最受欢迎的5家奶茶店、中式快餐的位置编号；

In [108]:
```
#求出最受欢迎的得分
merge_data_max['high_score_popular']=((merge_data_max['score'].astype('int')/5)*0.7)+((merge_data_max['per_pay']/merge_data_max['high_pay'])*0.3)
merge_data_max
```

Out[108]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_nai
0	14	深圳	862	7	1	4	2	美
1	51	北京	659	12	2	3	2	美
2	78	上海	699	8	1	1	0	美
3	79	上海	1033	15	2	3	1	美
4	93	深圳	813	5	0	1	0	美
...	...	...	...	...	...	...	...	
126	1930	上海	993	7	2	2	2	美
127	1931	深圳	1109	4	0	4	0	美
128	1971	上海	642	6	4	4	0	美
129	1974	上海	1127	6	4	3	0	美
130	1997	上海	924	10	1	2	0	美

131 rows × 12 columns

In [109]:
```
#最受欢迎的5家奶茶店、中式快餐的商家
popular_shop=merge_data_max.sort_values(by=['city_name','cate_3_name','high_score_popular'],ascending=False)
top5_popular_shop=popular_shop.groupby(['city_name','cate_3_name']).head(5)
top5_popular_shop[['shop_id','city_name','cate_3_name','high_score_popular']]
```

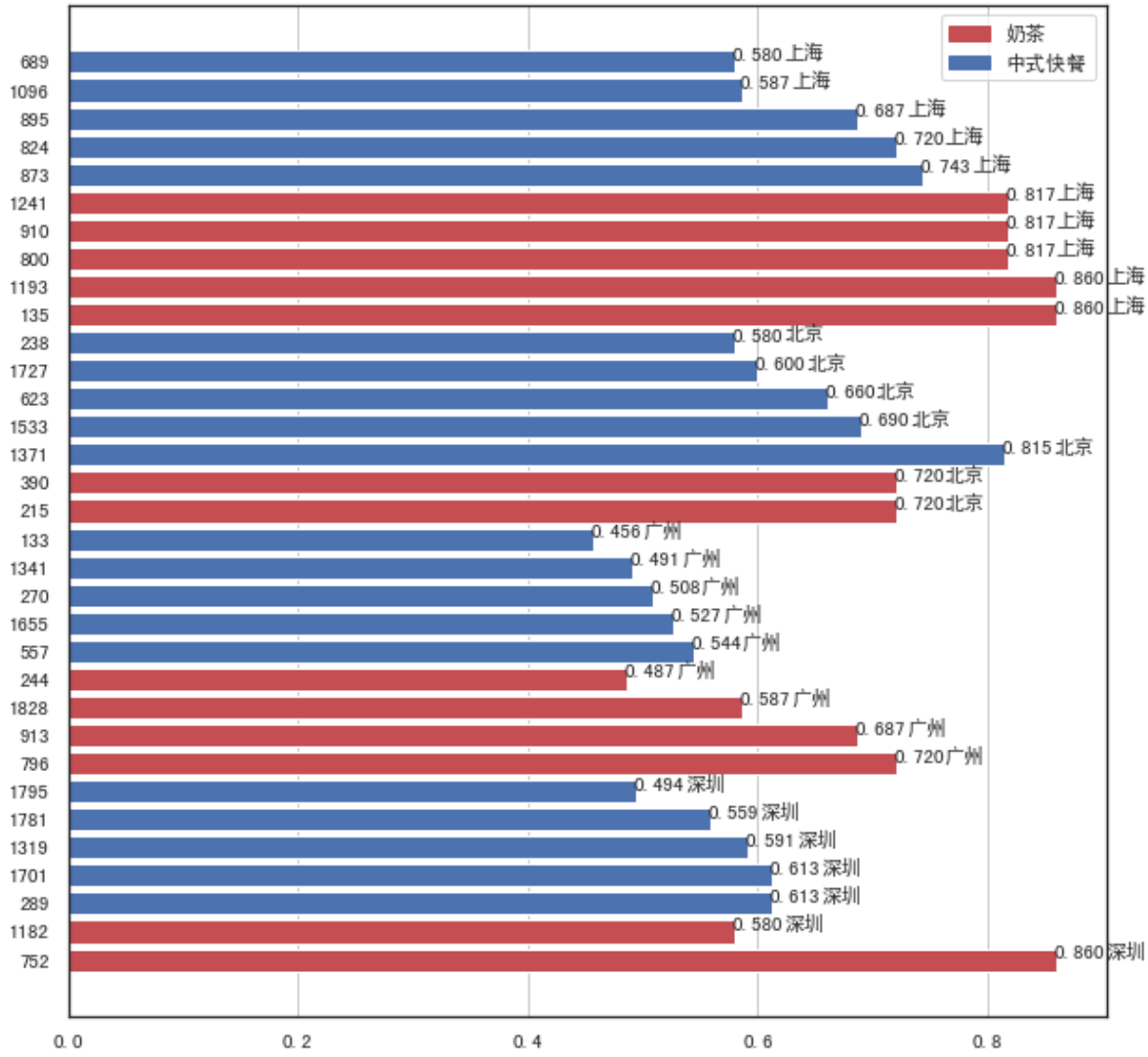
Out[109]:

	shop_id	city_name	cate_3_name	high_score_popular
49	752	深圳	奶茶	0.860000
75	1182	深圳	奶茶	0.580000
22	289	深圳	中式快餐	0.612857
107	1701	深圳	中式快餐	0.612857
81	1319	深圳	中式快餐	0.591429
114	1781	深圳	中式快餐	0.558571
115	1795	深圳	中式快餐	0.494286

53	796	广州	奶茶	0.720000
63	913	广州	奶茶	0.686667
117	1828	广州	奶茶	0.586667
16	244	广州	奶茶	0.486667
36	557	广州	中式快餐	0.543529
102	1655	广州	中式快餐	0.527059
20	270	广州	中式快餐	0.508235
83	1341	广州	中式快餐	0.490588
8	133	广州	中式快餐	0.456471
13	215	北京	奶茶	0.720000
28	390	北京	奶茶	0.720000
85	1371	北京	中式快餐	0.815000
93	1533	北京	中式快餐	0.690000
41	623	北京	中式快餐	0.660000
110	1727	北京	中式快餐	0.600000
15	238	北京	中式快餐	0.580000
9	135	上海	奶茶	0.860000
76	1193	上海	奶茶	0.860000
55	800	上海	奶茶	0.817143
62	910	上海	奶茶	0.817143
79	1241	上海	奶茶	0.817143
59	873	上海	中式快餐	0.743333
57	824	上海	中式快餐	0.720000
60	895	上海	中式快餐	0.686667
73	1096	上海	中式快餐	0.586667
45	689	上海	中式快餐	0.580000

In [110]:
```
#用于tableau可视化文件
top5_popular_shop.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau可视化excel文件/北上广深四个城市最受欢迎的5家奶茶店、中式快餐以及评分（水平条形图）.xlsx')
```

In [112]:
```
#可视化
fig,ax1=plt.subplots(figsize=(10,10))
y1=top5_popular_shop['shop_id'].astype('str')
x=top5_popular_shop['high_score_popular'].values
y2=top5_popular_shop['city_name']
cate_3_name_color={'奶茶':'r','中式快餐':'b'}
for a,b,c in zip(x,y1,y2):
    plt.text(a+0.02,b,'%.3f'%a,ha='center',va='bottom',fontsize=11)
#设置条形图上的得分标签
    plt.text(a+0.06,b,c,ha='center',va='bottom',fontsize=11)#设置条形图上的城市标签
ax1.barh(y1,x,color=top5_popular_shop.cate_3_name.replace(cate_3_name_color))
legend_handles = [mpatches.Patch(color=color, label=cate_3_name)#此处为自定义图例
                  for cate_3_name, color in cate_3_name_color.items()]
ax1.legend(handles=legend_handles)
ax1.grid(axis='x')
plt.show()
```



以城市为单位，统计每个城市总体消费金额 (饼状图):付款次数*人均消费

In [ ]:
```
#1.计算每个商家的付款次数
```

In [506]:
```
time_stamp_count=user_pay[['shop_id','time_stamp']].groupby('shop_id',as_index=False).agg('count').rename(columns={'time_stamp':'payment_count'})
```

In [507]:
```
data_merge3=pd.merge(shop_info,time_stamp_count,left_on='shop_id',right_on='shop_id')
data_merge3
```

Out[507]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_na
0	1	湖州	885	8	4	12	2	
1	2	哈尔滨	64	19	NaN	NaN	1	超市便
2	3	南昌	774	5	3	2	0	
3	4	天津	380	18	NaN	NaN	1	超市便
4	5	杭州	263	2	2	2	0	
...	...	...	...	...	...	...	...	
1995	1996	南宁	248	6	3	1	0	
1996	1997	上海	924	10	1	2	0	
1997	1998	南通	1090	1	2	2	0	
1998	1999	成都	1134	19	NaN	NaN	1	超市便
1999	2000	杭州	378	7	3	2	0	

2000 rows × 11 columns

In [508]:
```
data_merge3['total_money']=data_merge3['per_pay']*data_merge3['payment_count']
data_merge3
```

Out[508]:

	shop_id	city_name	location_id	per_pay	score	comment_cnt	shop_level	cate_1_na
0	1	湖州	885	8	4	12	2	
1	2	哈尔滨	64	19	NaN	NaN	1	超市便
2	3	南昌	774	5	3	2	0	
3	4	天津	380	18	NaN	NaN	1	超市便
4	5	杭州	263	2	2	2	0	
...	...	...	...	...	...	...	...	
1995	1996	南宁	248	6	3	1	0	
1996	1997	上海	924	10	1	2	0	
1997	1998	南通	1090	1	2	2	0	
1998	1999	成都	1134	19	NaN	NaN	1	超市便
1999	2000	杭州	378	7	3	2	0	

2000 rows × 12 columns

In [436]:
```
# pivot_merge=data_merge3.pivot_table(index='city_name',values='tot
al_money',aggfunc='sum')
# pivot_merge
```

Out[436]:

city_name	total_money
三亚	382928
三明	382592
上海	118558484
上饶	302580
东莞	7498328
...	...
青岛	4728503
黄冈	848905
黄山	1052915
黄石	1427485
龙岩	127320

122 rows × 1 columns

In [510]:
```
pivot_merge=data_merge3[['city_name','total_money']].groupby('city_name',as_index=False).agg(sum)
pivot_merge
```

Out[510]:

	city_name	total_money
0	三亚	382928
1	三明	382592
2	上海	118558484
3	上饶	302580
4	东莞	7498328
...	...	...
117	青岛	4728503
118	黄冈	848905
119	黄山	1052915
120	黄石	1427485
121	龙岩	127320

122 rows × 2 columns

In [511]:
```
#算出各个城市的总消费金额以及占比
money_total=pivot_merge['total_money'].sum()
pivot_merge['city_percent']=pivot_merge.apply(lambda x:x['total_money']/money_total,axis=1)
sort_merge=pivot_merge.sort_values(by='city_percent',ascending=False)
sort_merge
```

Out[511]:

	city_name	total_money	city_percent
2	上海	118558484	0.146744
53	杭州	104997921	0.129960
13	北京	54783654	0.067808
91	苏州	48044007	0.059466
15	南京	46676819	0.057774
...	...	...	...
67	淮北	113465	0.000140
44	德阳	112640	0.000139
42	张家口	109801	0.000136
56	梧州	106100	0.000131
41	廊坊	52576	0.000065

122 rows × 3 columns

In [517]:
```
df1=sort_merge.iloc[:10,]#筛选出前十消费金额的城市
df1
```

Out[517]:

	city_name	total_money	city_percent
2	上海	118558484	0.146744
53	杭州	104997921	0.129960
13	北京	54783654	0.067808
91	苏州	48044007	0.059466
15	南京	46676819	0.057774
40	广州	43977860	0.054433
57	武汉	38522742	0.047681
69	深圳	30470475	0.037714
34	宁波	23170496	0.028679
71	温州	20047564	0.024814

In [522]:
```
sort_merge.iloc[10:,:1]='其他'#将前十之外的城市名称都赋值成'其他'
df2=sort_merge.iloc[10:,:]
df3=df2.groupby('city_name',as_index=False).agg(sum)#算出其他城市的总消费金额以及比例
df3
```

Out[522]:

	city_name	total_money	city_percent
0	其他	278675598	0.344927

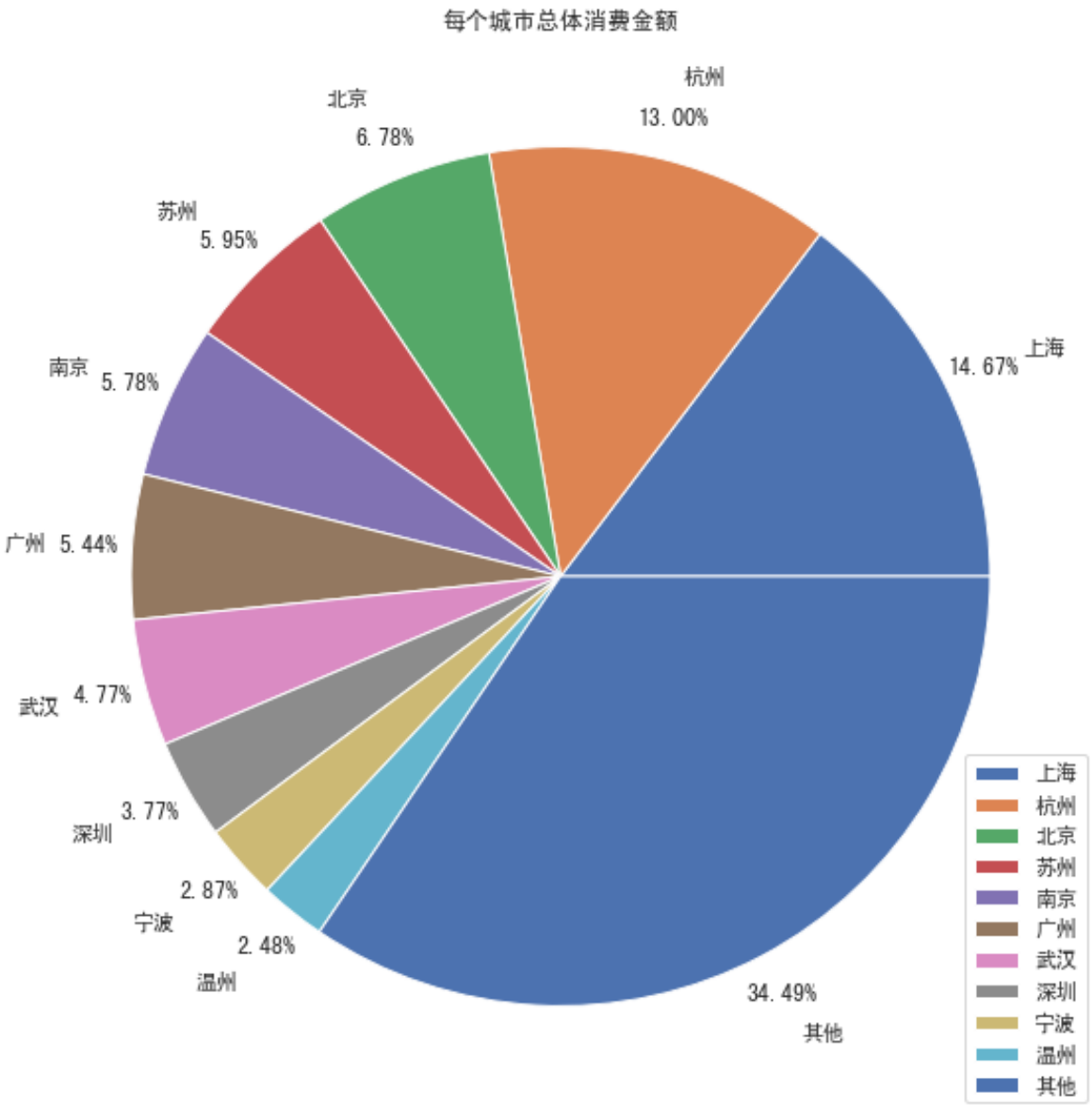
In [528]:
```
#合并前十城市与其他城市
merge_df=pd.concat([df1,df3])
merge_df
```

Out[528]:

	city_name	total_money	city_percent
2	上海	118558484	0.146744
53	杭州	104997921	0.129960
13	北京	54783654	0.067808
91	苏州	48044007	0.059466
15	南京	46676819	0.057774
40	广州	43977860	0.054433
57	武汉	38522742	0.047681
69	深圳	30470475	0.037714
34	宁波	23170496	0.028679
71	温州	20047564	0.024814
0	其他	278675598	0.344927

In [730]:
```
#用于tableau可视化文件
merge_df.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau 可视化excel文件/城市总体消费占比（饼图）.xlsx')
```

In [628]:
```
data=merge_df['city_percent']
labels=merge_df['city_name']
plt.figure(figsize=(10,10))
plt.pie(data,labels=labels,autopct='%.2f%%',pctdistance=1.1,labeldistance=1.2)
plt.title('每个城市总体消费金额')
plt.legend(loc='lower right')
plt.show()
```



以天为单位，统计所有商家交易发生次数和被用户浏览次数 （曲线图）

计算每天所有商家交易发生次数

In [6]:
```
#转换time_stamp时间格式，提取日期格式
user_pay['time_stamp']=pd.to_datetime(user_pay['time_stamp'])
user_pay['day_stamp']=user_pay['time_stamp'].dt.date
```

In [7]:
```
#算出每日的商家交易次数
group1=user_pay[['day_stamp','time_stamp']].groupby('day_stamp',as_
index=False).agg('count').rename(columns={'time_stamp':'payment_cou
nt'})
group1
```

Out[7]:

	day_stamp	payment_count
0	2015-06-26	66
1	2015-06-27	106
2	2015-06-28	146
3	2015-06-29	492
4	2015-06-30	598
...	...	...
488	2016-10-27	278055
489	2016-10-28	300797
490	2016-10-29	321920
491	2016-10-30	317073
492	2016-10-31	270748

493 rows × 2 columns

计算每天用户浏览次数

In [9]:
```
user_view['time_stamp']=pd.to_datetime(user_view['time_stamp'])
user_view['day_stamp']=user_view['time_stamp'].dt.date
```

In [10]:
```
#算出每日的用户浏览次数次数
group2=user_view[['day_stamp','time_stamp']].groupby('day_stamp',as_index=False).agg('count').rename(columns={'time_stamp':'view_count','day_stamp':'date_stamp'})
group2
```

Out[10]:

	date_stamp	view_count
0	2016-06-22	42244
1	2016-06-23	63589
2	2016-06-24	71587
3	2016-06-25	111912
4	2016-06-26	51091
...	...	...
125	2016-10-27	32052
126	2016-10-28	32023
127	2016-10-29	43921
128	2016-10-30	44056
129	2016-10-31	39549

130 rows × 2 columns

In [12]:
```
#每天所有商家交易发生次数的初始日期与每天用户浏览次数初始日期不匹配需用concat合并表
concat1=pd.concat([group1,group2])
concat1
```

Out[12]:

	day_stamp	payment_count	date_stamp	view_count
0	2015-06-26	66.0	NaN	NaN
1	2015-06-27	106.0	NaN	NaN
2	2015-06-28	146.0	NaN	NaN
3	2015-06-29	492.0	NaN	NaN
4	2015-06-30	598.0	NaN	NaN
...	...	...	...	...
125	NaN	NaN	2016-10-27	32052.0
126	NaN	NaN	2016-10-28	32023.0
127	NaN	NaN	2016-10-29	43921.0
128	NaN	NaN	2016-10-30	44056.0
129	NaN	NaN	2016-10-31	39549.0

623 rows × 4 columns

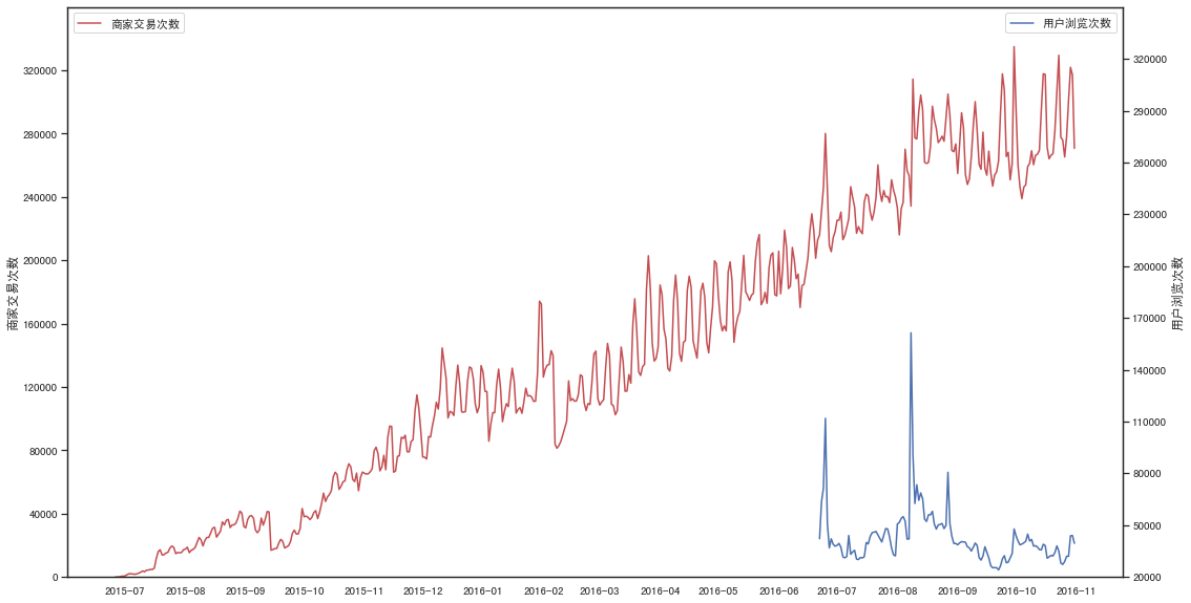
In [15]:
```
concat1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 623 entries, 0 to 129
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   day_stamp      493 non-null    object 
 1   payment_count  493 non-null    float64
 2   date_stamp     130 non-null    object 
 3   view_count     130 non-null    float64
dtypes: float64(2), object(2)
memory usage: 24.3+ KB
```

In [16]:
```
#将object格式转换成datetime格式以便可视化识别
concat1['day_stamp']=pd.to_datetime(concat1['day_stamp'])
concat1['date_stamp']=pd.to_datetime(concat1['date_stamp'])
```

In [21]:
```
#用于tableau可视化文件
concat1.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau 可视化excel文件/所有商家每日交易发生次数和每日用户浏览次数（曲线图）.xlsx')
```

In [20]:
```
fig,ax1=plt.subplots(figsize=(18,10))
x1=concat1['day_stamp']#此处必须为datetime格式否则xaxis设置每月定位符和日期格式无法识别
x2=concat1['date_stamp']#此处必须为datetime格式否则xaxis设置每月定位符和日期格式无法识别
y1=concat1['payment_count']
y2=concat1['view_count']
ax1.plot(x1,y1,color='r',label='商家交易次数')
ax2=ax1.twinx()
ax2.plot(x2,y2,color='b',label='用户浏览次数')
ax1.set_ylabel('商家交易次数')
ax2.set_ylabel('用户浏览次数')
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
#设置每月定位符
ax1.xaxis.set_major_locator(mdates.MonthLocator()) # interval = 1
#设置日期的格式
ax1.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
#设置y轴上的刻度
ax1.set_ylim([0,360000])#y1轴数值范围
ax1.set_yticks(range(0,360000,40000))#设置y1轴的刻度范围 range（启始值，末值，每个刻度的跨度）
ax1.set_yticklabels(range(0,360000,40000))#设置y1轴上的刻度
ax2.set_ylim([20000,350000])#y2轴数值范围
ax2.set_yticks(range(20000,350000,30000))#设置y2轴的刻度范围
ax2.set_yticklabels(range(20000,350000,30000))#设置2轴上的刻度
plt.savefig("所有商家交易发生次数和被用户浏览次数曲线图.pdf",bbox_inche='tight')#将图片保存为pdf格式，将其设置为“tight”可以正确保存所保存的图形
plt.show()
```



留存分析。对于平均日交易额最大的前3个商家，对他们进行漏斗分析，以浏览行为作为分析目标，输出2016.10.01~2016.10.31 共31天的留存率

查找出“平均日交易额”最大的前3个商家

In [668]:
```
user_pay['time_stamp']=pd.to_datetime(user_pay['time_stamp'])
user_pay['ymd']=user_pay['time_stamp'].dt.date
user_pay
```

Out[668]:

	user_id	shop_id	time_stamp	day_stamp	month_stamp	date_stamp	ymd
0	22127870	1862	2015-12-25 17:00:00	2015-12-25	12	2015-12-25	2015-12-25
1	3434231	1862	2016-10-05 11:00:00	2016-10-05	10	2016-10-05	2016-10-05
2	16955285	1862	2016-02-10 15:00:00	2016-02-10	2	2016-02-10	2016-02-10
3	13799128	1862	2016-01-13 14:00:00	2016-01-13	1	2016-01-13	2016-01-13
4	13799128	1862	2016-07-05 12:00:00	2016-07-05	7	2016-07-05	2016-07-05
...	...	...	...	...	...	...	...
69674105	15330495	766	2016-08-21 19:00:00	2016-08-21	8	2016-08-21	2016-08-21
69674106	8556331	766	2016-08-11 20:00:00	2016-08-11	8	2016-08-11	2016-08-11
69674107	11429329	766	2016-09-18 18:00:00	2016-09-18	9	2016-09-18	2016-09-18
69674108	11429329	766	2016-10-02 20:00:00	2016-10-02	10	2016-10-02	2016-10-02
69674109	22602094	766	2016-10-13 13:00:00	2016-10-13	10	2016-10-13	2016-10-13

69674110 rows × 7 columns

In [669]:
```
user_pay.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69674110 entries, 0 to 69674109
Data columns (total 7 columns):
 #   Column       Dtype         
---  ------       -----         
 0   user_id      int32         
 1   shop_id      int16         
 2   time_stamp   datetime64[ns]
 3   day_stamp    datetime64[ns]
 4   month_stamp  int64         
 5   date_stamp   object        
 6   ymd          object        
dtypes: datetime64[ns](2), int16(1), int32(1), int64(1), object(2)
memory usage: 3.0+ GB
```

1.先计算每个shop_id的营收，人均消费*付款次数

In [671]:
```
user_pay
```

Out[671]:

	user_id	shop_id	time_stamp	day_stamp	month_stamp	date_stamp	ymd
0	22127870	1862	2015-12-25 17:00:00	2015-12-25	12	2015-12-25	2015-12-25
1	3434231	1862	2016-10-05 11:00:00	2016-10-05	10	2016-10-05	2016-10-05
2	16955285	1862	2016-02-10 15:00:00	2016-02-10	2	2016-02-10	2016-02-10
3	13799128	1862	2016-01-13 14:00:00	2016-01-13	1	2016-01-13	2016-01-13
4	13799128	1862	2016-07-05 12:00:00	2016-07-05	7	2016-07-05	2016-07-05
...	...	...	...	...	...	...	...
69674105	15330495	766	2016-08-21 19:00:00	2016-08-21	8	2016-08-21	2016-08-21
69674106	8556331	766	2016-08-11 20:00:00	2016-08-11	8	2016-08-11	2016-08-11
69674107	11429329	766	2016-09-18 18:00:00	2016-09-18	9	2016-09-18	2016-09-18
69674108	11429329	766	2016-10-02 20:00:00	2016-10-02	10	2016-10-02	2016-10-02
69674109	22602094	766	2016-10-13 13:00:00	2016-10-13	10	2016-10-13	2016-10-13

69674110 rows × 7 columns

In [652]:
```
#计算付款次数
group_paymentcount=user_pay[['shop_id','time_stamp']].groupby('shop_id').agg('count').rename(columns={'time_stamp':'payment_count'})
group_paymentcount
```

Out[652]:

shop_id	payment_count
1	92689
2	39110
3	10469
4	13606
5	31455
...	...
1996	7313
1997	40774
1998	45021
1999	95661
2000	20248

2000 rows × 1 columns

In [672]:
```
#每个店的总营收
merge_shopid=pd.merge(shop_info,group_paymentcount,left_on ='shop_id',right_on='shop_id')
merge_shopid['total_revenue']=merge_shopid['per_pay']*merge_shopid['payment_count']
total_shopid=merge_shopid[['shop_id','total_revenue']].groupby('shop_id').agg('sum')
total_shopid
```

Out[672]:

shop_id	total_revenue
1	741512
2	743090
3	52345
4	244908
5	62910
...	...
1996	43878
1997	407740
1998	45021
1999	1817559
2000	141736

2000 rows × 1 columns

2.计算每个shop_id的开店时长，最近开店时间-初始开店时间

In [683]:
```
#首次付款时间为初始开店时间
shopid_firstpay=user_pay[['shop_id','time_stamp']].groupby('shop_id
').agg('min').rename(columns={'time_stamp':'firt_pay'})
#最近付款时间为最近开店时间
shopid_lastpay=user_pay[['shop_id','time_stamp']].groupby('shop_id'
).agg('max').rename(columns={'time_stamp':'last_pay'})
merge_pay=pd.merge(shopid_lastpay,shopid_firstpay,left_on='shop_id'
,right_on='shop_id')
merge_pay
```

Out[683]:

shop_id	last_pay	firt_pay
1	2016-10-31 23:00:00	2015-10-10 15:00:00
2	2016-10-31 21:00:00	2015-11-25 19:00:00
3	2016-10-31 21:00:00	2016-06-18 10:00:00
4	2016-10-31 21:00:00	2016-07-19 13:00:00
5	2016-10-31 22:00:00	2015-09-28 13:00:00
...	...	...
1996	2016-10-31 23:00:00	2016-07-19 11:00:00
1997	2016-10-31 20:00:00	2015-10-28 11:00:00
1998	2016-10-31 23:00:00	2015-11-02 10:00:00
1999	2016-10-31 22:00:00	2015-12-05 12:00:00
2000	2016-10-31 20:00:00	2016-05-06 10:00:00

2000 rows × 2 columns

In [684]:
```
#算出opening_time为开店时长
merge_pay['firt_pay']=merge_pay['firt_pay'].dt.date
merge_pay['last_pay']=merge_pay['last_pay'].dt.date
merge_pay['opening_time']=merge_pay['last_pay']-merge_pay['firt_pay
']
```

In [675]:
```
merge_pay
```

Out[675]:

shop_id	last_pay	firt_pay	opening_time
1	2016-10-31	2015-10-10	387 days
2	2016-10-31	2015-11-25	341 days
3	2016-10-31	2016-06-18	135 days
4	2016-10-31	2016-07-19	104 days
5	2016-10-31	2015-09-28	399 days
...	...	...	...
1996	2016-10-31	2016-07-19	104 days
1997	2016-10-31	2015-10-28	369 days
1998	2016-10-31	2015-11-02	364 days
1999	2016-10-31	2015-12-05	331 days
2000	2016-10-31	2016-05-06	178 days

2000 rows × 3 columns

3.计算日均消费前3的商家

In [696]:
```
merge_top=pd.merge(total_shopid,merge_pay,left_on='shop_id',right_on='shop_id')
merge_top
```

Out[696]:

shop_id	total_revenue	last_pay	firt_pay	opening_time
1	741512	2016-10-31	2015-10-10	387 days
2	743090	2016-10-31	2015-11-25	341 days
3	52345	2016-10-31	2016-06-18	135 days
4	244908	2016-10-31	2016-07-19	104 days
5	62910	2016-10-31	2015-09-28	399 days
...	...	...	...	...
1996	43878	2016-10-31	2016-07-19	104 days
1997	407740	2016-10-31	2015-10-28	369 days
1998	45021	2016-10-31	2015-11-02	364 days
1999	1817559	2016-10-31	2015-12-05	331 days
2000	141736	2016-10-31	2016-05-06	178 days

2000 rows × 4 columns

In [697]:
```
merge_top.info()#需要将opening_time的timedelta格式转换成整数形式才能进行数学运算
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000 entries, 1 to 2000
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype          
---  ------         --------------  -----          
 0   total_revenue  2000 non-null   int64          
 1   last_pay       2000 non-null   object         
 2   firt_pay       2000 non-null   object         
 3   opening_time   2000 non-null   timedelta64[ns]
dtypes: int64(1), object(2), timedelta64[ns](1)
memory usage: 78.1+ KB
```

In [699]:
```
#将opening_time的天数提取出来,去掉days
merge_top['opening_time']=merge_top['opening_time'].dt.days
```

In [701]:
```
merge_top.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000 entries, 1 to 2000
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   total_revenue   2000 non-null   int64 
 1   last_pay        2000 non-null   object
 2   firt_pay        2000 non-null   object
 3   opening_time    2000 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 78.1+ KB
```

让opening_time变成整数的思路2：首先转换成字符串格式，然后利用apply和匿名函数取里面每个元素中的数字部分，去掉后四位[:-5]，然后字符串之间转换成int32格式即可

merge_top['opening_time']=merge_top['opening_time'].astype('str').apply(lambda x:x[:-5]).astype('int32')

In [702]:
```
merge_top
```

Out[702]:

shop_id	total_revenue	last_pay	firt_pay	opening_time
1	741512	2016-10-31	2015-10-10	387
2	743090	2016-10-31	2015-11-25	341
3	52345	2016-10-31	2016-06-18	135
4	244908	2016-10-31	2016-07-19	104
5	62910	2016-10-31	2015-09-28	399
...	...	...	...	...
1996	43878	2016-10-31	2016-07-19	104
1997	407740	2016-10-31	2015-10-28	369
1998	45021	2016-10-31	2015-11-02	364
1999	1817559	2016-10-31	2015-12-05	331
2000	141736	2016-10-31	2016-05-06	178

2000 rows × 4 columns

In [704]:
```
#计算每家shop_id日均消费情况
merge_top['avg_dailyrevenue']=merge_top['total_revenue']/merge_top['opening_time']
merge_top
```

Out[704]:

shop_id	total_revenue	last_pay	firt_pay	opening_time	avg_dailyrevenue
1	741512	2016-10-31	2015-10-10	387	1916.051680
2	743090	2016-10-31	2015-11-25	341	2179.149560
3	52345	2016-10-31	2016-06-18	135	387.740741
4	244908	2016-10-31	2016-07-19	104	2354.884615
5	62910	2016-10-31	2015-09-28	399	157.669173
...	...	...	...	...	...
1996	43878	2016-10-31	2016-07-19	104	421.903846
1997	407740	2016-10-31	2015-10-28	369	1104.986450
1998	45021	2016-10-31	2015-11-02	364	123.684066
1999	1817559	2016-10-31	2015-12-05	331	5491.114804
2000	141736	2016-10-31	2016-05-06	178	796.269663

2000 rows × 5 columns

In [705]:
```
#日均消费前三的商家
merge_top.sort_values(by='avg_dailyrevenue',ascending=False).head(3)
```

Out[705]:

shop_id	total_revenue	last_pay	firt_pay	opening_time	avg_dailyrevenue
1629	13609280	2016-10-31	2015-11-11	355	38336.000000
517	6463128	2016-10-31	2016-04-27	187	34562.181818
58	9047971	2016-10-31	2015-10-30	367	24653.871935

以上得知日均消费前三的商家shop_id为：1629，517，58

对以上日均消费前三的商家进行留存分析

In [708]:
```
user_view['time_stamp']=pd.to_datetime(user_view['time_stamp'])
user_view['ymd']=user_view['time_stamp'].dt.date
user_view
```

Out[708]:

	user_id	shop_id	time_stamp	day_stamp	ymd
0	13201967	1197	2016-10-21 18:00:00	2016-10-21	2016-10-21
1	19461365	1197	2016-06-28 23:00:00	2016-06-28	2016-06-28
2	15022321	1197	2016-07-16 19:00:00	2016-07-16	2016-07-16
3	5440872	1197	2016-07-15 07:00:00	2016-07-15	2016-07-15
4	12594529	1197	2016-08-07 16:00:00	2016-08-07	2016-08-07
...	...	...	...	...	...
5556710	3426068	590	2016-08-26 21:00:00	2016-08-26	2016-08-26
5556711	2519370	590	2016-10-29 13:00:00	2016-10-29	2016-10-29
5556712	7978950	590	2016-09-24 08:00:00	2016-09-24	2016-09-24
5556713	13699038	590	2016-10-30 14:00:00	2016-10-30	2016-10-30
5556714	12426548	590	2016-10-03 11:00:00	2016-10-03	2016-10-03

5556715 rows × 5 columns

In [713]:
```
#将ymd的字符串格式转换成datetime格式才能进行条件筛选
user_view['ymd']=pd.to_datetime(user_view['ymd'])
```

In [714]:
```
#挑选出日期为 2016.10.01~2016.10.31 , shop_id为1629, 517, 58的浏览量
select_top3=user_view.query("shop_id in [1629,517,58] & ymd >= '2016-10-01' & ymd <= '2016-10-31'")#query函数条件筛选需要双引号里写条件，日期需引号
select_top3
```

Out[714]:

	user_id	shop_id	time_stamp	day_stamp	ymd
2275032	10071392	517	2016-10-22 21:00:00	2016-10-22	2016-10-22
2275043	9541932	517	2016-10-18 21:00:00	2016-10-18	2016-10-18
2275050	8584386	517	2016-10-13 12:00:00	2016-10-13	2016-10-13
2275059	19453077	517	2016-10-21 10:00:00	2016-10-21	2016-10-21
2275061	19926664	517	2016-10-29 20:00:00	2016-10-29	2016-10-29
...	...	...	...	...	...
4372918	3417021	58	2016-10-28 22:00:00	2016-10-28	2016-10-28
4372919	3417021	58	2016-10-28 22:00:00	2016-10-28	2016-10-28
4372947	10940335	58	2016-10-20 07:00:00	2016-10-20	2016-10-20
4372980	20705153	58	2016-10-20 22:00:00	2016-10-20	2016-10-20
4372981	20705153	58	2016-10-25 17:00:00	2016-10-25	2016-10-25

5147 rows × 5 columns

In [715]:
```
#1.计算每个用户user_id最早浏览时间
select_top3['min_day']=select_top3.groupby('user_id')['ymd'].transform(min)
select_top3.head()
```

Out[715]:

	user_id	shop_id	time_stamp	day_stamp	ymd	min_day
2275032	10071392	517	2016-10-22 21:00:00	2016-10-22	2016-10-22	2016-10-22
2275043	9541932	517	2016-10-18 21:00:00	2016-10-18	2016-10-18	2016-10-18
2275050	8584386	517	2016-10-13 12:00:00	2016-10-13	2016-10-13	2016-10-13
2275059	19453077	517	2016-10-21 10:00:00	2016-10-21	2016-10-21	2016-10-21
2275061	19926664	517	2016-10-29 20:00:00	2016-10-29	2016-10-29	2016-10-29

In [716]:
```
#2.计算每个用户浏览日ymd和最早访问时间min_day的时间间隔
select_top3["day_gap"] = (((select_top3["ymd"] - select_top3["min_day"])//1) + dt.timedelta(days=1)).apply(lambda x:x.days)#对已知日期进行增减计算用timedelta
#日期间隔已经计算出来，但后面带有一个单位 days，这是因为两个 datetime 类型相减，得到的数据类型是 timedelta64，如果只要数字，还需要使用 timedelta 的 days 属性转换一下。
#用lambda函数把求出day_gap数据后面的day去掉
select_top3
```

Out[716]:

	user_id	shop_id	time_stamp	day_stamp	ymd	min_day	day_gap
2275032	10071392	517	2016-10-22 21:00:00	2016-10-22	2016-10-22	2016-10-22	1
2275043	9541932	517	2016-10-18 21:00:00	2016-10-18	2016-10-18	2016-10-18	1
2275050	8584386	517	2016-10-13 12:00:00	2016-10-13	2016-10-13	2016-10-13	1
2275059	19453077	517	2016-10-21 10:00:00	2016-10-21	2016-10-21	2016-10-21	1
2275061	19926664	517	2016-10-29 20:00:00	2016-10-29	2016-10-29	2016-10-29	1
...	...	...	...	...	...	...	...
4372918	3417021	58	2016-10-28 22:00:00	2016-10-28	2016-10-28	2016-10-28	1
4372919	3417021	58	2016-10-28 22:00:00	2016-10-28	2016-10-28	2016-10-28	1
4372947	10940335	58	2016-10-20 07:00:00	2016-10-20	2016-10-20	2016-10-20	1
4372980	20705153	58	2016-10-20 22:00:00	2016-10-20	2016-10-20	2016-10-20	1
4372981	20705153	58	2016-10-25 17:00:00	2016-10-25	2016-10-25	2016-10-20	6

5147 rows × 7 columns

In [717]:
```
#3.按照首次访问时间和时间间隔来统计用户数
#按照首次访问时间和下一次访问的间隔，统计用户数,用户user_id去重统计每个组的用户数,需注意要pd.Series调取nunique的函数
group_top3 = select_top3.groupby(["min_day","day_gap"])["user_id"].agg(pd.Series.nunique).reset_index()
group_top3
```

Out[717]:

	min_day	day_gap	user_id
0	2016-10-01	1	244
1	2016-10-01	2	2
2	2016-10-01	3	2
3	2016-10-01	4	2
4	2016-10-01	6	2
...	...	...	...
230	2016-10-29	2	2
231	2016-10-29	3	2
232	2016-10-30	1	125
233	2016-10-30	2	1
234	2016-10-31	1	129

235 rows × 3 columns

In [718]:
```
#4.生成数据透视表
pivot_top3 = group_top3.pivot_table(index="min_day",columns="day_gap",values="user_id")
pivot_top3
```

Out[718]:

day_gap	1	2	3	4	5	6	7	8	9	10	...	18	19	20
min_day														
2016-10-01	244.0	2.0	2.0	2.0	NaN	2.0	2.0	1.0	NaN	3.0	...	1.0	NaN	1.0
2016-10-02	189.0	4.0	3.0	1.0	3.0	2.0	3.0	NaN	NaN	4.0	...	NaN	2.0	NaN
2016-10-03	147.0	2.0	1.0	1.0	1.0	NaN	2.0	NaN	1.0	1.0	...	1.0	NaN	NaN
2016-10-04	126.0	2.0	1.0	NaN	1.0	2.0	NaN	1.0	NaN	NaN	...	1.0	1.0	1.0
2016-10-05	135.0	1.0	NaN	NaN	NaN	NaN	NaN	1.0	NaN	1.0	...	NaN	NaN	NaN
2016-10-06	143.0	3.0	NaN	3.0	2.0	NaN	1.0	1.0	1.0	2.0	...	1.0	NaN	NaN

2016-10-07	184.0	2.0	2.0	1.0	NaN	2.0	NaN	NaN	2.0	2.0	...	1.0	NaN	NaN	N
2016-10-08	173.0	3.0	NaN	1.0	1.0	1.0	1.0	NaN	NaN	NaN	...	NaN	1.0	NaN	N
2016-10-09	165.0	2.0	1.0	NaN	1.0	1.0	1.0	NaN	1.0	1.0	...	NaN	NaN	NaN	N
2016-10-10	176.0	3.0	NaN	NaN	NaN	2.0	1.0	NaN	1.0	NaN	...	1.0	1.0	NaN	
2016-10-11	123.0	NaN	NaN	1.0	3.0	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	
2016-10-12	141.0	1.0	NaN	1.0	1.0	NaN	NaN	NaN	1.0	1.0	...	NaN	1.0	1.0	N
2016-10-13	151.0	2.0	4.0	NaN	1.0	1.0	NaN	1.0	NaN	NaN	...	NaN	1.0	NaN	N
2016-10-14	120.0	2.0	1.0	NaN	NaN	1.0	NaN	NaN	2.0	NaN	...	NaN	NaN	NaN	N
2016-10-15	160.0	NaN	1.0	1.0	NaN	1.0	NaN	2.0	NaN	1.0	...	NaN	NaN	NaN	N
2016-10-16	162.0	2.0	NaN	NaN	1.0	NaN	1.0	2.0	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-17	114.0	1.0	NaN	1.0	NaN	2.0	1.0	NaN	1.0	NaN	...	NaN	NaN	NaN	N
2016-10-18	125.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-19	112.0	NaN	NaN	1.0	1.0	2.0	1.0	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-20	116.0	NaN	1.0	NaN	1.0	2.0	1.0	1.0	1.0	NaN	...	NaN	NaN	NaN	N
2016-10-21	101.0	NaN	1.0	NaN	NaN	NaN	1.0	1.0	1.0	NaN	...	NaN	NaN	NaN	N
2016-10-22	170.0	1.0	1.0	1.0	NaN	1.0	2.0	1.0	3.0	NaN	...	NaN	NaN	NaN	N
2016-10-23	129.0	NaN	NaN	2.0	2.0	1.0	NaN	2.0	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-24	68.0	1.0	1.0	NaN	NaN	2.0	NaN	1.0	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-25	75.0	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-26	66.0	2.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-27	82.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-	103.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N

10-28															
2016-10-29	153.0	2.0	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-30	125.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
2016-10-31	129.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N

31 rows × 27 columns

In [719]:
```
#5.改变数据形式：以百分比显示数据
#取出第一列的数据（全部行）
size=pivot_top3.iloc[:,0]
size.head()
```

Out[719]:
```
min_day
2016-10-01    244.0
2016-10-02    189.0
2016-10-03    147.0
2016-10-04    126.0
2016-10-05    135.0
Name: 1, dtype: float64
```

In [720]:
```
#后面每个day_gap数据除以相应日期的第一个数据:使用divide函数
final=pivot_top3.divide(size,axis=0).round(3)  # 在行的方向上除以对应size中的值并保留3为小数
final.head()
```

Out[720]:

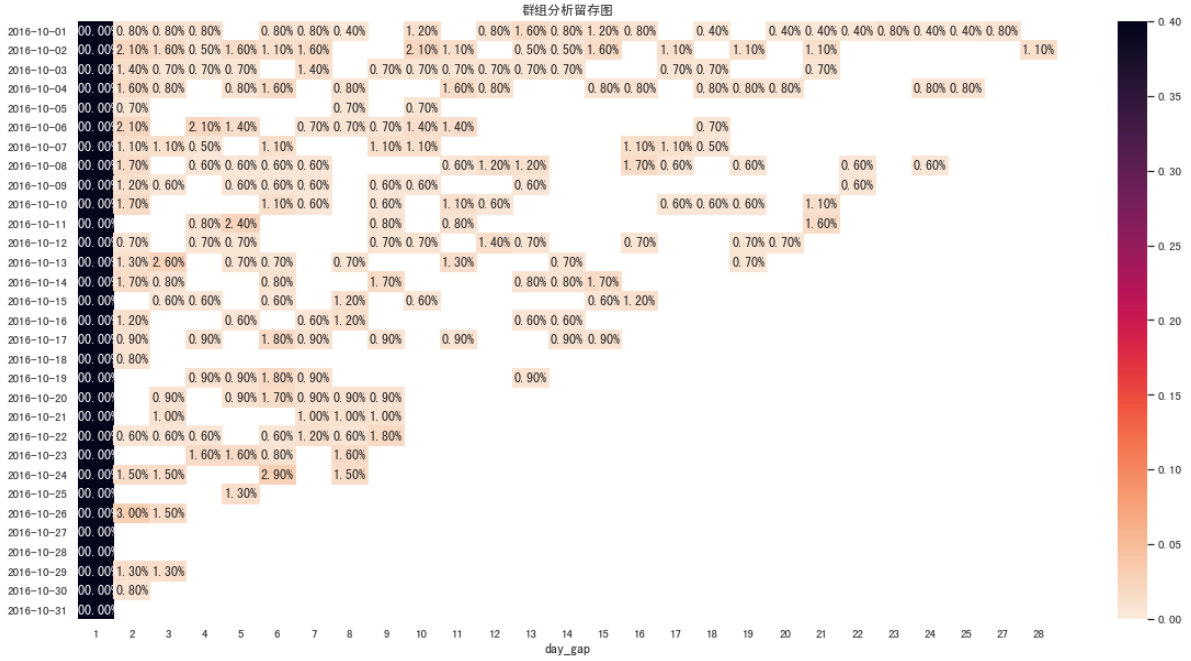
day_gap	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**...**	**18**	**19**
min_day													
2016-10-01	1.0	0.008	0.008	0.008	NaN	0.008	0.008	0.004	NaN	0.012	...	0.004	NaN
2016-10-02	1.0	0.021	0.016	0.005	0.016	0.011	0.016	NaN	NaN	0.021	...	NaN	0.01
2016-10-03	1.0	0.014	0.007	0.007	0.007	NaN	0.014	NaN	0.007	0.007	...	0.007	NaN
2016-10-04	1.0	0.016	0.008	NaN	0.008	0.016	NaN	0.008	NaN	NaN	...	0.008	0.008
2016-10-05	1.0	0.007	NaN	NaN	NaN	NaN	NaN	0.007	NaN	0.007	...	NaN	NaN

5 rows × 27 columns

In [721]:
```
# 索引重置，只取出年月日，后面的时分秒取消
final.index = final.index.date
```

In [750]:
```
#绘制留存分析热力图
plt.figure(figsize=(20,10))
plt.title("群组分析留存图")

sns.heatmap(data=final,
            annot=True,#显示每个方格的数据标签
            fmt='.2%',#保留的后面2位小数
            vmin = 0.0,#vmax和vmax显示右侧的最大值和最小值的颜色柱
            vmax = 0.4,
            cmap="rocket_r"#热力图的颜色
           )
plt.show()
```



In [747]:
```
#用于tableau可视化文件
select_top3.to_excel('./Desktop/阿里巴巴口碑商家流量分析/tableau 可视化excel文件/平均日交易额前三3个商家用户留存分析（热力图）.xlsx')
```