

北京大学信息科学技术学院

外卖数据分析

以饿了么为例的外卖数据分析

聂鹏飞、薛犇、杨天猛、陈家园

2018-06-23

目录

1. 背景意义	1
1. 已有工作	1
2. 存在问题	1
3. 技术挑战	1
4. 工作目标及意义	2
2. 实验原理	2
1. 开发环境	2
2. 实验设计	2
3. 实验内容	3
1. 技术方案	3
4. 实验结果	7
1. 参数与数据	7
2. 分析结果	8

1. 背景意义

1. 已有工作

Python2 环境下的urllib\urllib2 爬虫样例代码

饿了么API:

餐厅分布API

菜单信息API

餐厅评分API

品类分布API

geohash获取API

2. 存在问题

1. 现有API不能提供充分所需的信息
2. 由于爬虫依据地理位置获取餐馆信息，需要爬取不同城市的poi(place of interest)
3. 爬虫效率较低，不能实现多次实时爬取
4. 爬取过程存在封IP、封账号问题

3. 技术挑战

1. 如何获得指定地理范围内的poi
2. 如何监控部分移动端API接口
3. 如何加速爬取速度。为保证所爬取数据的时效性，需要在尽可能短的时间里（至多30分钟）爬取完毕北京市2万余家餐厅的近300万菜品
4. 如何应对反爬机制
5. 如何获得较好的可视化效果
6. 如何对菜品名称进行关键词提取

4. 工作目标及意义

以饿了么外卖数据为参考，分析城市间外送餐饮行业发展差异；了解外送餐饮在地理上或时间上的分布差异以做出优化调整；对未来订单的预测可为商家提供指导；对商品促销、活动的数据分析可以为商家选择合理折扣比例、提升商品利润提供参考。

2. 实验原理

1. 开发环境

系统环境：Win10、OS X

编程环境：Python3.6

依赖库：urllib.request、selenium、requests

2. 实验设计

1. 利用poi提取器获取不同城市（北上广南杭）城区的均匀分布的poi
2. 从这些poi出发，爬取覆盖范围内所有的餐厅基础信息(包含ID/名称/分类/菜单/月售/评分/评价等)
3. 每天的不同时间点(01:00/14:00/20:00)根据餐厅ID，遍历爬取餐厅所有商品的相关信息(包含ID/名称/月售/原价/折扣价/库存剩余/评分/满意度等)
4. 对这两部分些信息进行分析、可视化

3. 实验内容

1. 技术方案

1. 基础餐厅信息(包括餐厅ID数据集)的获取方式

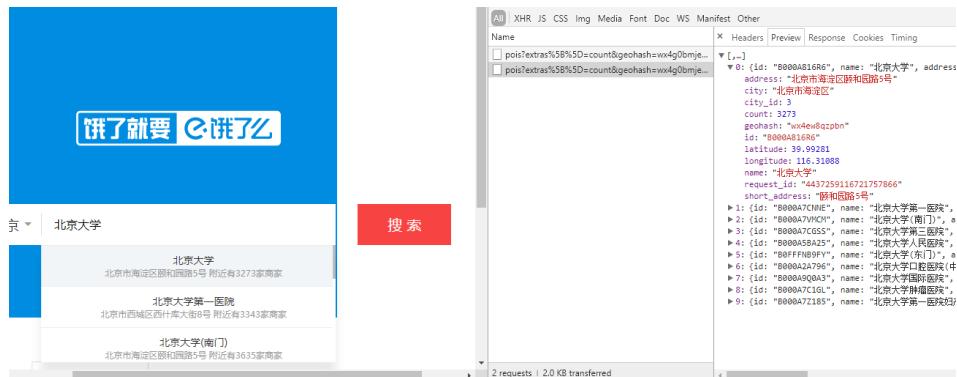
为了自动爬取指定城市的poi，我们研究了饿了么的定位系统，发现了如下的API：

```
https://www.ele.me/restAPI/v2/pois?  
extras%5B%5D=count&geohash=wx4g0bmjetr7&keyword=%E5%8C%97%E4%BA%AC&li  
mit=20&type=nearby
```

这个API有如下几个输入参数选项：

- 1.Count
- 2.Geohash
- 3.Keyword
- 4.Limit

它的返回值是满足输入条件的一系列poi。也就是说，我们可以根据输入的地理位置，以及关键词，得到某个位置附近的poi。如图所示：



观察一下输入，其中最关键的是geohash和keyword。Geohash就是对经纬度的一种hash算法，将二维的经纬度转换成字符串，比如下图展示了北京9个区域的GeoHash字符串，分别是WX4ER，WX4G2、WX4G3等等，每一个字符串代表了某一矩形区域。也就是说，这个矩形区域内所有的点（经纬度坐标）都共享相同的GeoHash字符串，这样既可以保护隐私（只表示大概区域位置而不是具体的点），又比较容易做缓存，比如左上角这个区域内的用户不断发送位置信息请求餐馆数据，由于这

些用户的GeoHash字符串都是WX4ER，所以可以把WX4ER当作key，把该区域的餐馆信息当作value来进行缓存，而如果不使用GeoHash的话，由于区域内的用户传来的经纬度是各不相同的，很难做缓存。



而 key
字符串，我们在
的时候并不需要。

word就是一些中文字
根据地理位置找poi

因此我们的设计思路如下，给定一个区域范围，我们就在这个范围内均匀地取一些点，由于这些点可能不是在饿了么数据库中的poi，所以，我们就计算出这些点的geohash，再传入获得poi的API中，取返回值中的第一个，也就是离该点最近的poi，作为爬得的poi。

之后我们就可以利用爬取餐厅信息的API来检索餐厅。具体的逻辑是不断增大API中的offset值，直到爬完，或者达到一个阈值。这个offset值在实际的网页中对应的就是“点击此处，加载下一页”，我们实际上是在利用程序，模拟人不断点击下一页的动作，这样就可以爬取更多的餐馆。

对于每一个餐馆，我们会得到这样的返回值：

```

[{"activities": [{"address": "广州市天河区新塘南约村口大街新塘综合市场粮油商铺22号", "auth_id": "6363170783448542", "description": "子丑黄泡(科学城店)", "distance": 3815, "favored": false, "flavors": [{"id": 209, "name": "盖浇饭"}, {"id": 265, "name": "简餐"}], "float_delivery_fee": 7.5, "float_minimum_order_amount": 20, "has_story": false, "id": 1447642, "image_path": "Sealf0e34e759b8d67a882e8290729f7.jpeg", "is_new": false, "is_premium": false, "is_stock_empty": 0, "is_valid": 1, "latitude": 23.166758, "longitude": 113.436903, "max_applied_quantity_per_order": -1, "name": "子丑黄泡(科学城店)", "next_business_time": "", "only_use_poi": false, "opening_hours": ["10:00/14:00"]}, {"activities": [{"address": "广州市天河区新塘南约村口大街新塘综合市场粮油商铺22号", "auth_id": "6363170783448542", "description": "子丑黄泡(科学城店)", "distance": 3815, "favored": false, "flavors": [{"id": 209, "name": "盖浇饭"}, {"id": 265, "name": "简餐"}], "float_delivery_fee": 7.5, "float_minimum_order_amount": 20, "has_story": false, "id": 1447642, "image_path": "Sealf0e34e759b8d67a882e8290729f7.jpeg", "is_new": false, "is_premium": false, "is_stock_empty": 0, "is_valid": 1, "latitude": 23.166758, "longitude": 113.436903, "max_applied_quantity_per_order": -1, "name": "子丑黄泡(科学城店)", "next_business_time": "", "only_use_poi": false, "opening_hours": ["10:00/14:00"]}]}

```

1 requests | 60.0 KB transferred | Finish: 431 ms | DO...

我们关心如下的几个变量：

1. 餐厅id
2. 名称 name
3. 经纬度 latitude, longitude
4. 种类 flavors
5. 日销量 recent_order_num

需要注意的是，从两个poi出发爬取到的餐厅很有可能有重复的部分，这很好理解，在北京大学和清华大学分别搜索附近商家，应该都能够搜到中关村的汉堡王，所以我们的程序还对餐厅id进行了去重。

2. 定时爬取商品菜单的获取方式

爬取全部菜单所使用的API:

https://www.ele.me/restapi/shopping/v2/menu?restaurant_id=2165785

API可选参数为restaurant_id，它的返回值为指定餐馆的全部商品信息。

在这部分信息中，我们仅保留其中的有用信息：

1. 商品标识信息：餐厅ID、商品ID、商品名等
2. 商品评价：评分、评分人数、满意度、满意度评分人数等
3. 价格与销售信息：原价、现价（折扣后价格）、月售（不常变动）、库存剩余（例：初始值9999，卖出一份则减一，每月清零，依此计算实时销售量）

通过网络请求遍历餐厅ID，访问API即可获取每家餐厅的全部商品目

录。一个商品ID只会出现在一家餐厅下，但可能重复出现在不同分区（例如在折扣分区与主食分区同时出现），所以需要去重。

3. 爬虫代码实现方式

爬取速度方面的改进与处理：

由于爬取餐厅基本信息耗时较短，重复需求不高，所以对速度、换IP代理、登陆状态等方面要求不高，初期可以使用基础的urllib库实现的逐条请求代码。

后来在爬取部分API时需要解决登陆状态与切换IP的问题，我们转而使用selenium库提供的调用chrome控制实现爬取（易于实现）。

与selenium易于实现相对应的是较低的效率与计算资源的部分浪费，使用selenium版本与初期的urllib版本爬取一个城市全部餐厅全部商品所需的时间约为1.5小时。而分析实时销售变化对数据的时效性要求较高，至少需要将爬取时间缩短至20分钟以内。

由于爬取餐厅信息的主要时间消耗在于网络响应等待，而非CPU计算。我们采取将多线程并发添加到爬虫代码中。最终利用threading库，实现了基于requests请求方式（这里requests库并不重要，只是出于易于编程方面而选择）的多线程（并行线程数限制在8000）爬虫。

多线程requests版本将原本的1.5小时爬取时间大幅缩短至约3分钟，使得每日分时段爬取工作可以继续进行。

应对反爬取方面的处理：

初期的逐条爬取过程没有遇到IP地址或账号方面的封禁，仅是部分API要求登陆状态。我们采取将保存登陆状态下cookie，添加到请求头中来完成反爬取。

中期逐条爬取过程开始遇到爬取时间过长时封禁IP的问题，采取爬取代理IP，通过代理服务器访问的方式解决。

在实现多线程爬取程序后，意外的发现反而不会遇到IP封禁。即使提高请求并行度（最多尝试到并行1.4w线程，此时已占满本机16G内存），也不会带来任何的IP封禁或账号封禁。猜测饿了么反爬取此时采取的是每隔一段时间检查恶意请求。

约无障碍爬取15天后，开始遇到快速封禁IP的状况。由于并行线程

数较大，IP封禁后，切换IP与重新发送请求的代价较大，效率大大降低，并且IP封禁速度过快，代理IP不够用切换。测试后猜测封禁策略是短时间内请求数限制。应对方式为降低并发线程数(降至十位数)，封禁时自动切换IP并重新请求。

4. 实验结果

1. 参数与数据

(1) 地理位置

我们选取了北京、上海、广州、南京、杭州5个城市进行分析，各个城市的经纬度选取如下：

北京：39.75~39.97, 116.22~116.75

上海：31.00~31.43, 121.26~121.91

广州：23.05~23.22, 113.26~113.46

南京：31.06~32.14, 118.61~119.02

杭州：30.09~30.36, 119.98~120.25

出于效率考虑，我们圈定的范围基本都不包含郊区，只分析城区内的外卖情况。

2. 数据集

- 内容：北京、上海、广州、南京、杭州的城区餐厅数据
位置：/数据集/北上广宁杭餐厅信息/（csv格式）
格式：序号/id/ latitude/ longitude/ flavor1/ flavor1_name/ flavor2/ flavor2_-name/ order_num/ avg_price
数量：每个城市1 - 2万条餐厅信息
爬取时间：2018-06-05
- 内容：北京、上海餐厅内所有菜品每日定时爬取的状态数据
位置：/数据集/每日菜单实时数据/（txt格式，逗号分隔）

格式: id/ food_id/ rating/ rating_count/ satisfy_count/ satisfy_rate/ original_price/ price/ promotion_stock/ recent_popularity/ recent_rating/ stock/month_sales

数量: 每个城市200-300万条菜品记录

爬取时间: 2018-05-31 - 2018-06-15 (每日01:00 / 14:00 / 20:00)

2. 分析结果

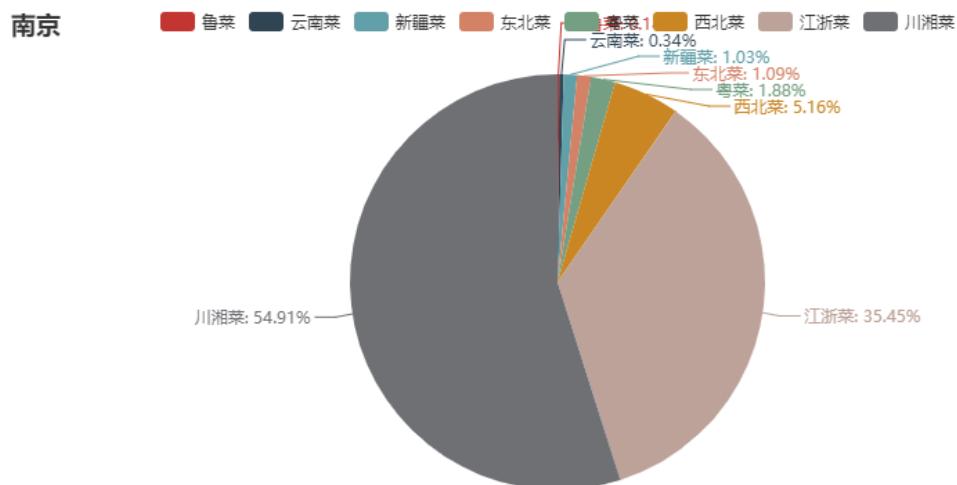
第一部分：菜品、菜系分布分析

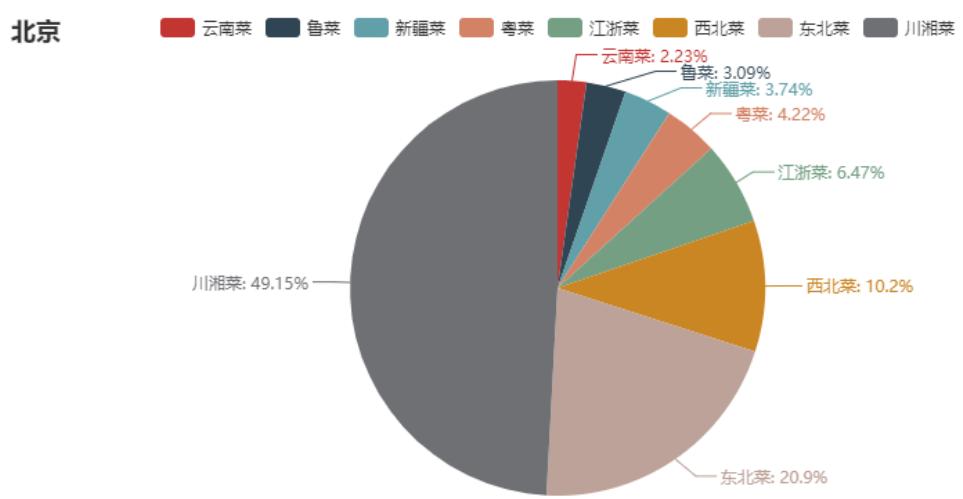
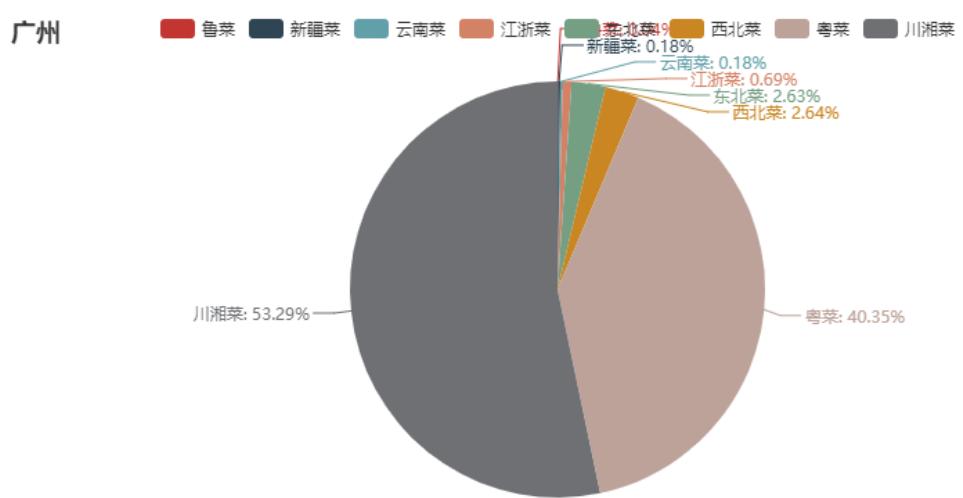
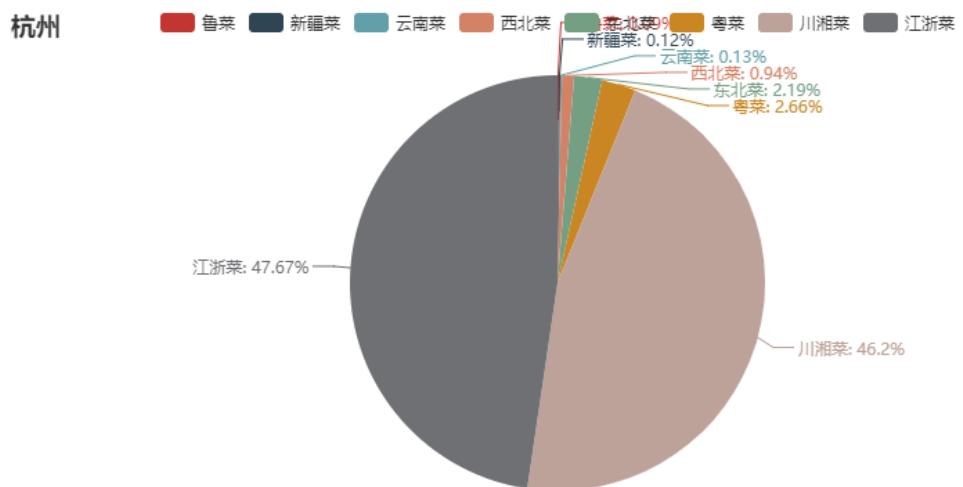
相关数据:

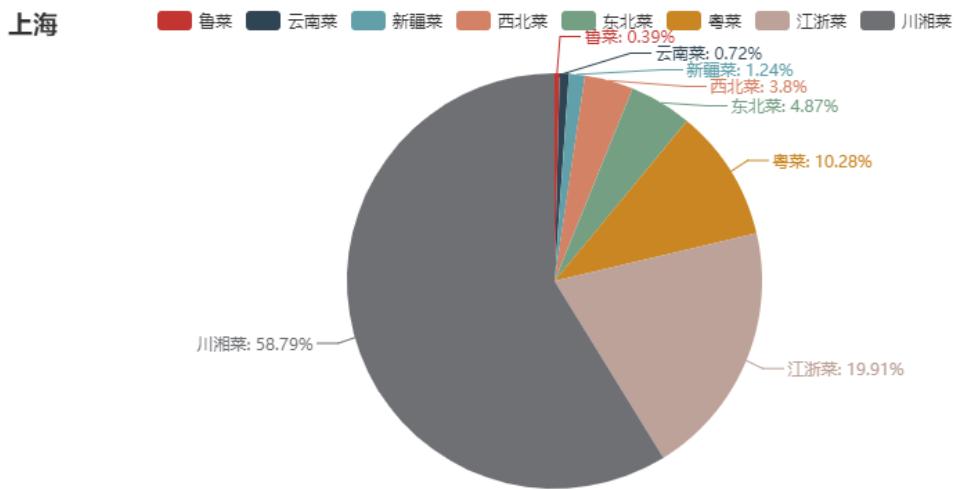
(一) 各城市菜系分布

我们按照销量占该城市所有菜系销量的比重，对于以下8大菜系在5个城市中的销售情况进行分析：

	云南菜	鲁菜	新疆菜	粤菜	江浙菜	西北菜	东北菜	川湘菜
北京	2.23%	3.09%	3.74%	4.22%	6.47%	10.2%	20.9%	49.2%
上海	0.72%	0.39%	1.24%	10.3%	19.9%	3.8%	4.87%	58.8%
广州	0.18%	0.04%	0.18%	40.3%	0.69%	2.64%	2.63%	53.3%
南京	0.34%	0.01%	1.03%	1.88%	35.5%	5.16%	1.09%	54.9%
杭州	0.13%	0.09%	0.12%	2.66%	47.7%	0.94%	2.19%	46.2%







从数据中我们可以总结出如下的结论：

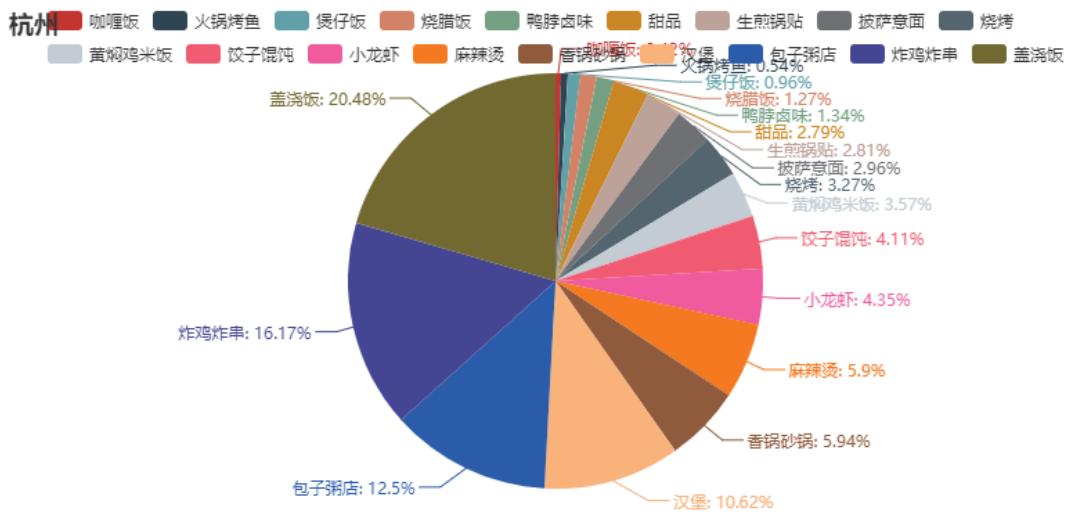
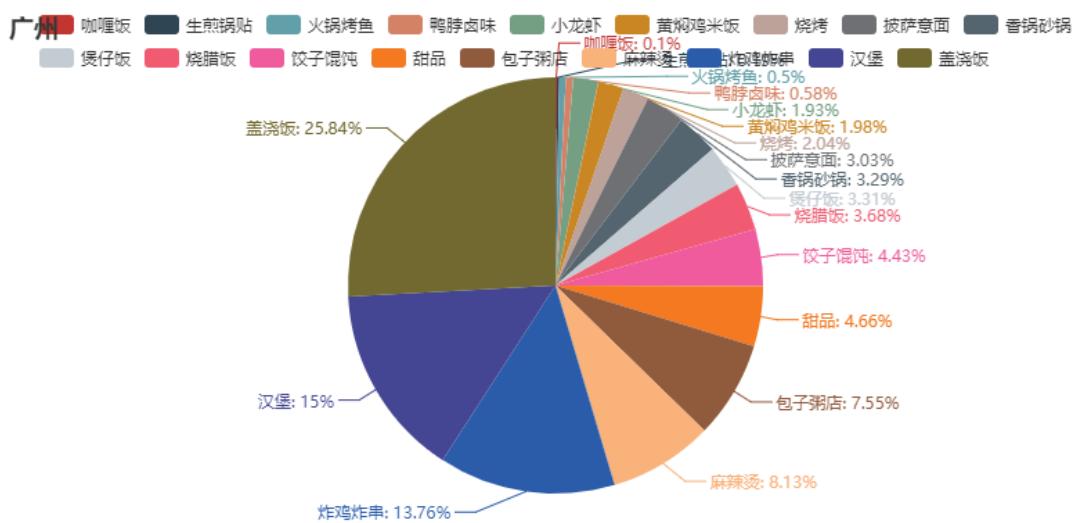
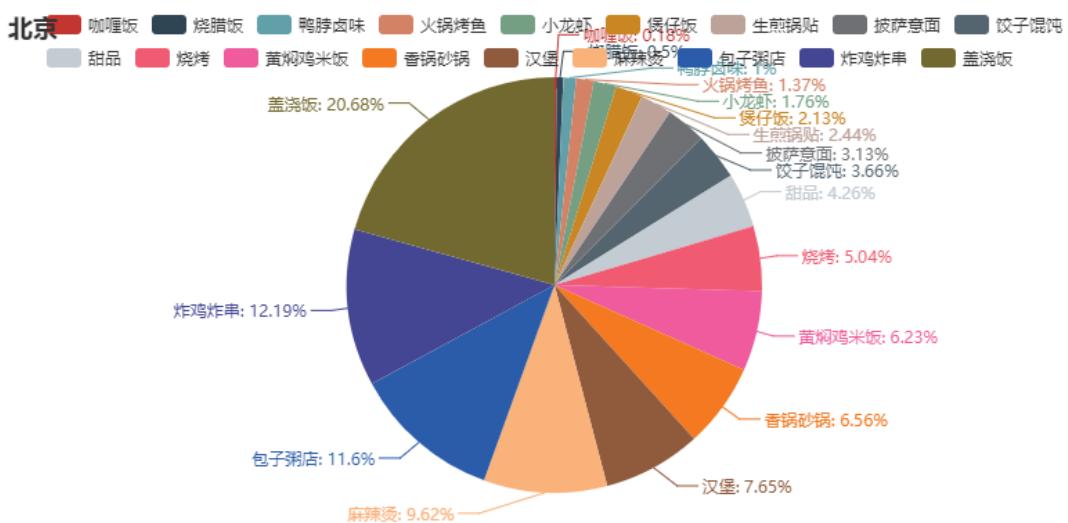
1. 菜系分布和地理位置有关，本帮菜较受欢迎。非常明显的，东北菜、西北菜在北京销量不错，江浙菜在沪宁杭销售量很好，粤菜在广州销售量非常好。
2. 川湘菜一家独大，如果想开一家饭店而不知道选什么口味的话，卖香辣的川湘菜估计生意不会差！川湘菜在各个大都市中都独占鳌头，统治了大家的口味。
3. 城市越大，菜系分布越均衡。以上海和南京这两个相近的城市做对比，除了川湘菜和江浙菜之外，剩余菜系在上海的比重要大于南京。同样也可以看到，中国最大的城市北京，它的菜系分布（除开川湘菜）是最均匀的。我们分析这是由于城市变大之后，外来人口也就增加，所以口味会更加杂。

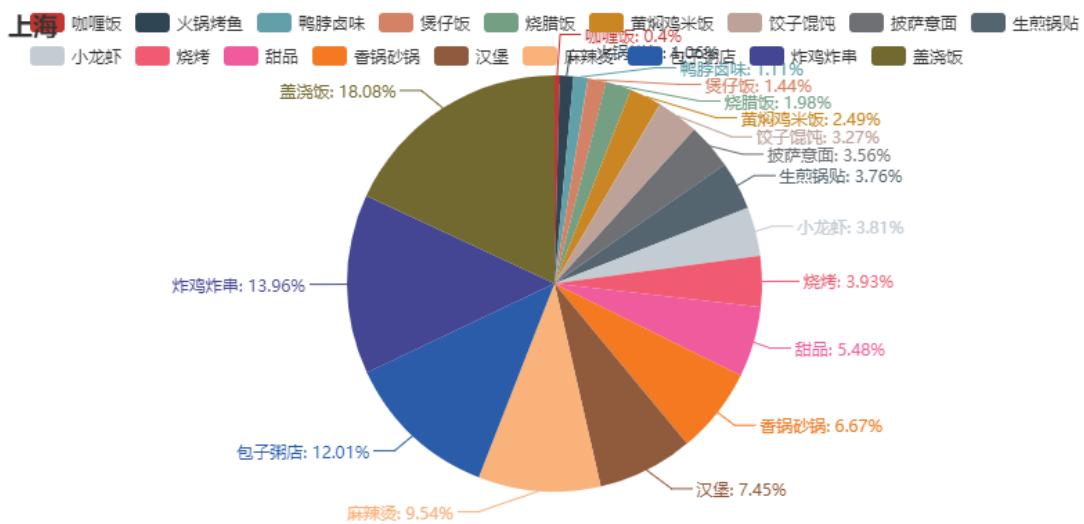
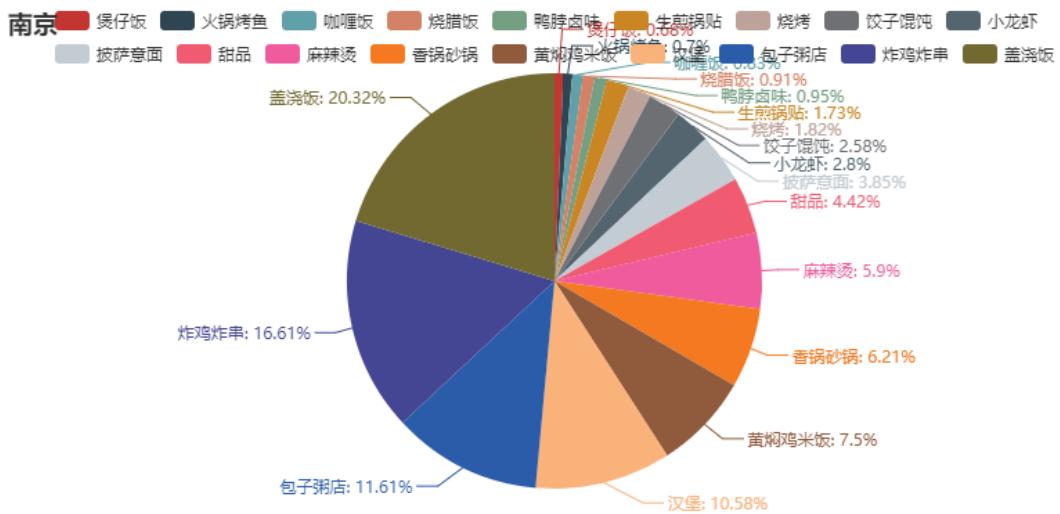
二. 各城市菜品种类分析

饿了么对餐馆除了按照菜系分类，还会按照菜品分类，有如下18种菜品：

煲仔饭，火锅烤鱼，咖喱饭，烧腊饭，鸭脖卤味，生煎锅贴，烧烤，饺子馄饨，小龙虾，披萨意面，甜品，麻辣烫，香锅砂锅，黄焖鸡米饭，汉堡，包子粥店，炸鸡炸串，盖浇饭。

我们按照这些菜品的销量，也进行了统计：

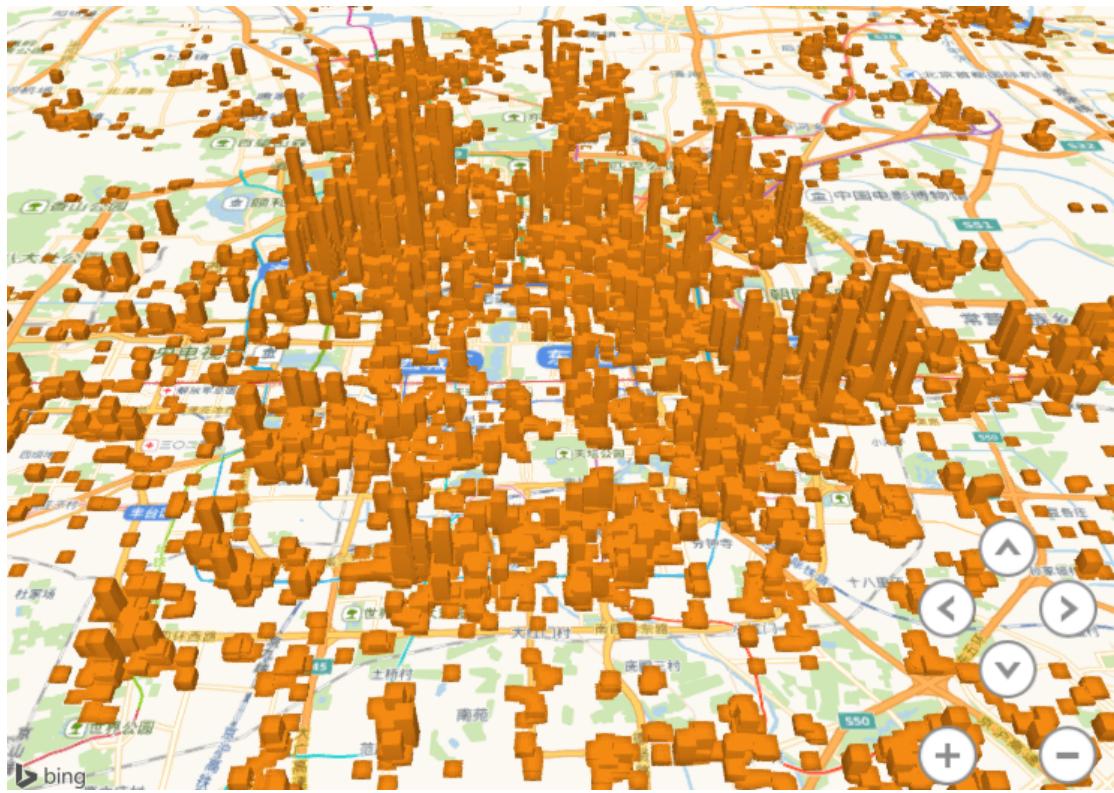




总结出的结论如下：

1. 盖浇饭这种中国快餐已经高居各地榜首了，其次是炸鸡炸串、汉堡这样的西式快餐，接着是包子粥店这样的中式小吃。可以看出，在大城市，人们更加青睐快餐。
2. 杭州人对于甜品不太感兴趣，可能是饭后更加喜欢喝龙井茶这样的高档茶点。
3. 南京人尤其不喜欢吃煲仔饭，但是很喜欢吃小龙虾。毕竟毗邻小龙虾之乡江苏盱眙。
4. 除了广州，烧腊饭的销量都很惨淡。只有广州人比较爱这种烧腊口味。
5. 上海人最喜欢吃生煎锅贴，毕竟生煎锅贴是有名的上海小吃。

三. 北京市外卖销售地区分布



可以看出，北京外卖销量最好的地方，就是海淀区，因为这里有着大量的大学和公司。这些地方的人大都比较忙碌，没有固定的时间做饭、吃饭，平时吃的食堂菜色也比较单一，所以对于外卖的需求量最高。

其次是东城区和朝阳区，这里的商圈比较繁华，所以也有很多经营店铺的业主，这些业主也没有可以回家的午饭时间，只能在工作地方解决。

销量最低的是二环靠西侧的西城区，可能因为这里大都是老北京人，没有吃外卖的习惯，或者是重要的政府机关，不能接触外卖，所以外卖的销量普遍很低。



如果把各个种类的菜品外卖进行可视化的话，我们可以发现也有细微的差别。像包子粥店这样的快餐分布和整体分布大致相当。但是像名酒坊这样的外卖就很有聚集性，比如都集中在东城区、朝阳区这样繁华的商圈，而海淀区这样的“外卖高销地区”就没有。

四. 菜系关键词提取

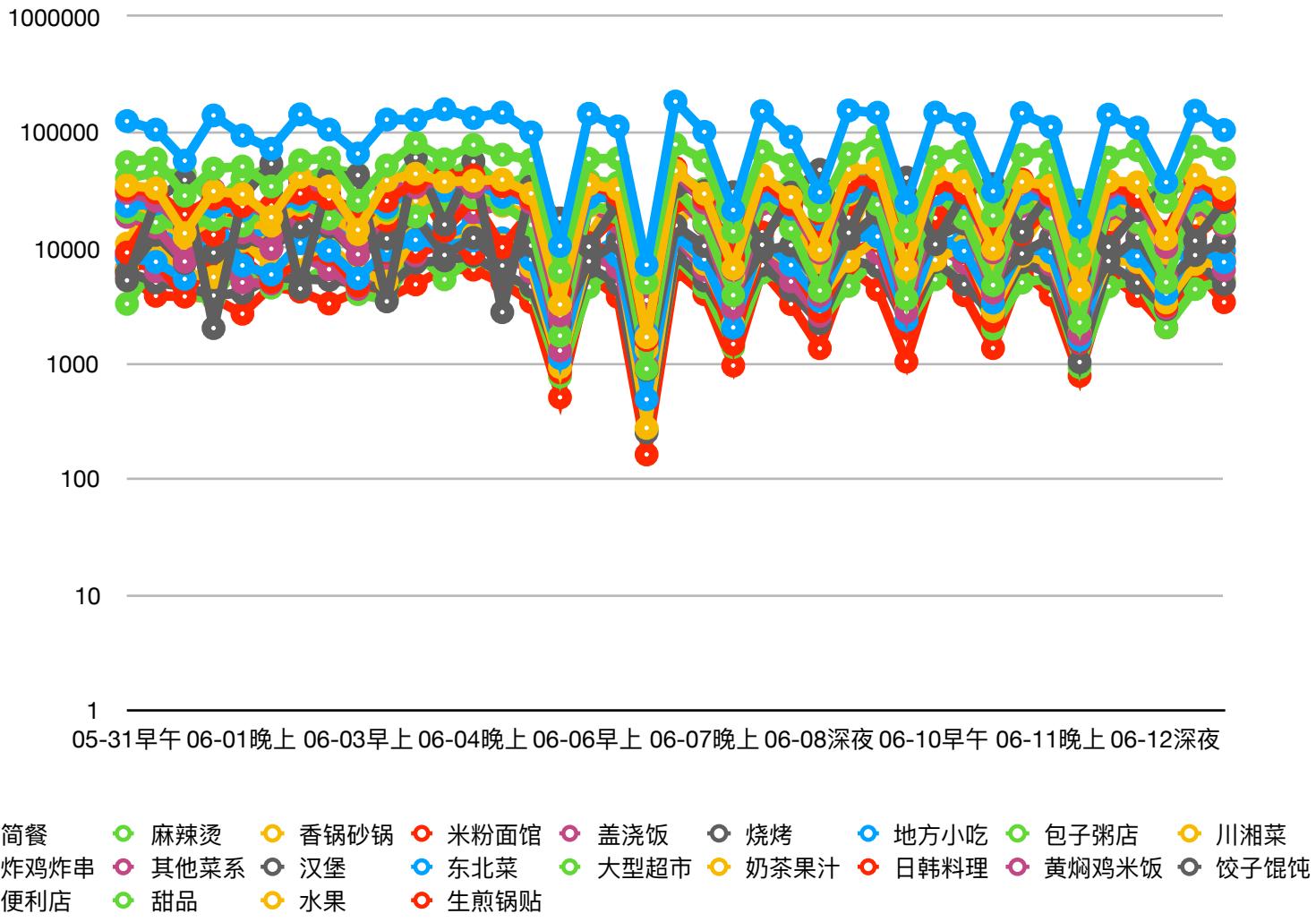
我们使用bigram对统计出来的菜品名称进行切词，川湘菜的结果如下：
川湘菜词频统计：

饭 0.02555313696856143
盖 0.018218389221061913
小 0.014440801256440636
干锅 0.012906070041054749
米饭 0.011473286859725016
份 0.010710054280439755
肉 0.010236133689692227
炒 0.009169812360510291
小炒 0.008891521781059708
套餐 0.008668338247044884
豆腐 0.00792163778139035
鸡蛋 0.007731518474636982
鸡 0.006714793486347229
牛肉 0.006571515168214256
鱼 0.006403438679635191
香 0.006353842338743009
肉丝 0.005483151020858017
西红柿 0.005372936929986498
土豆丝 0.005323340589094316
尖椒 0.0052764996004739
大 0.005171796214145979
麻辣 0.004940346623315791
豆角 0.004871462816521092
菜 0.004827377180172485

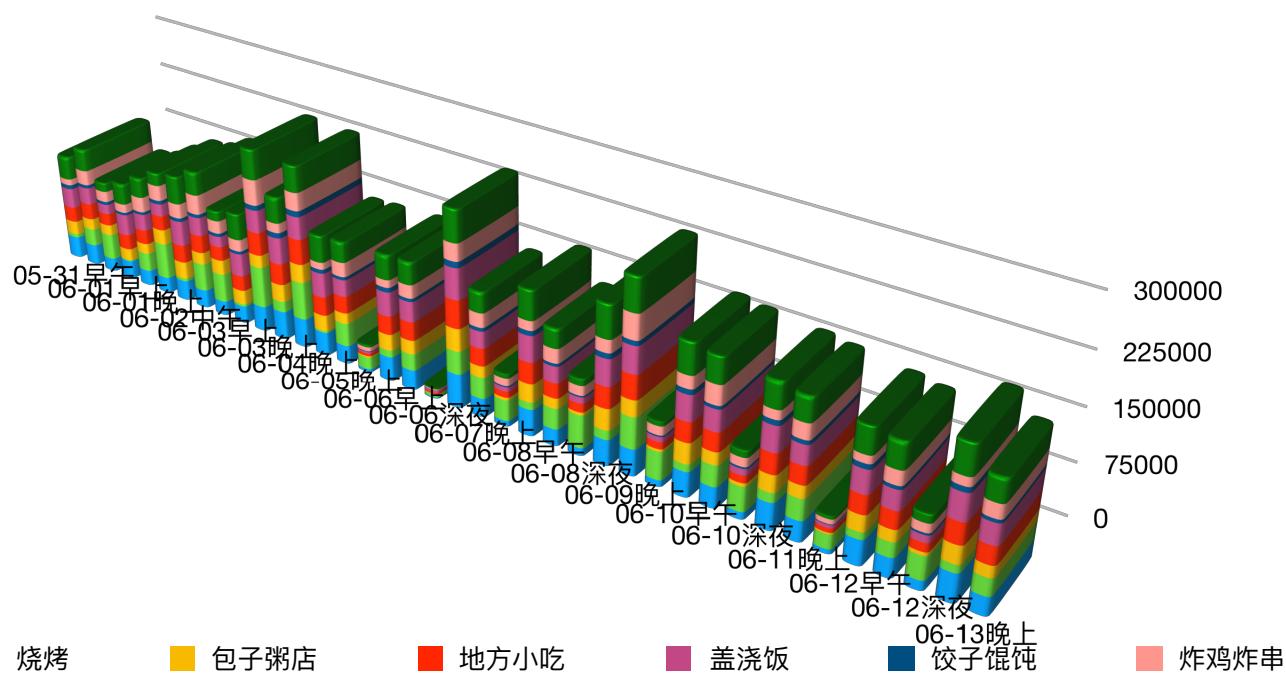
切出的词还是非常符合预取的，比如干锅、尖椒、麻辣等词的出现频率都很高。

第二部分：连续时间下不同菜式的销量变化

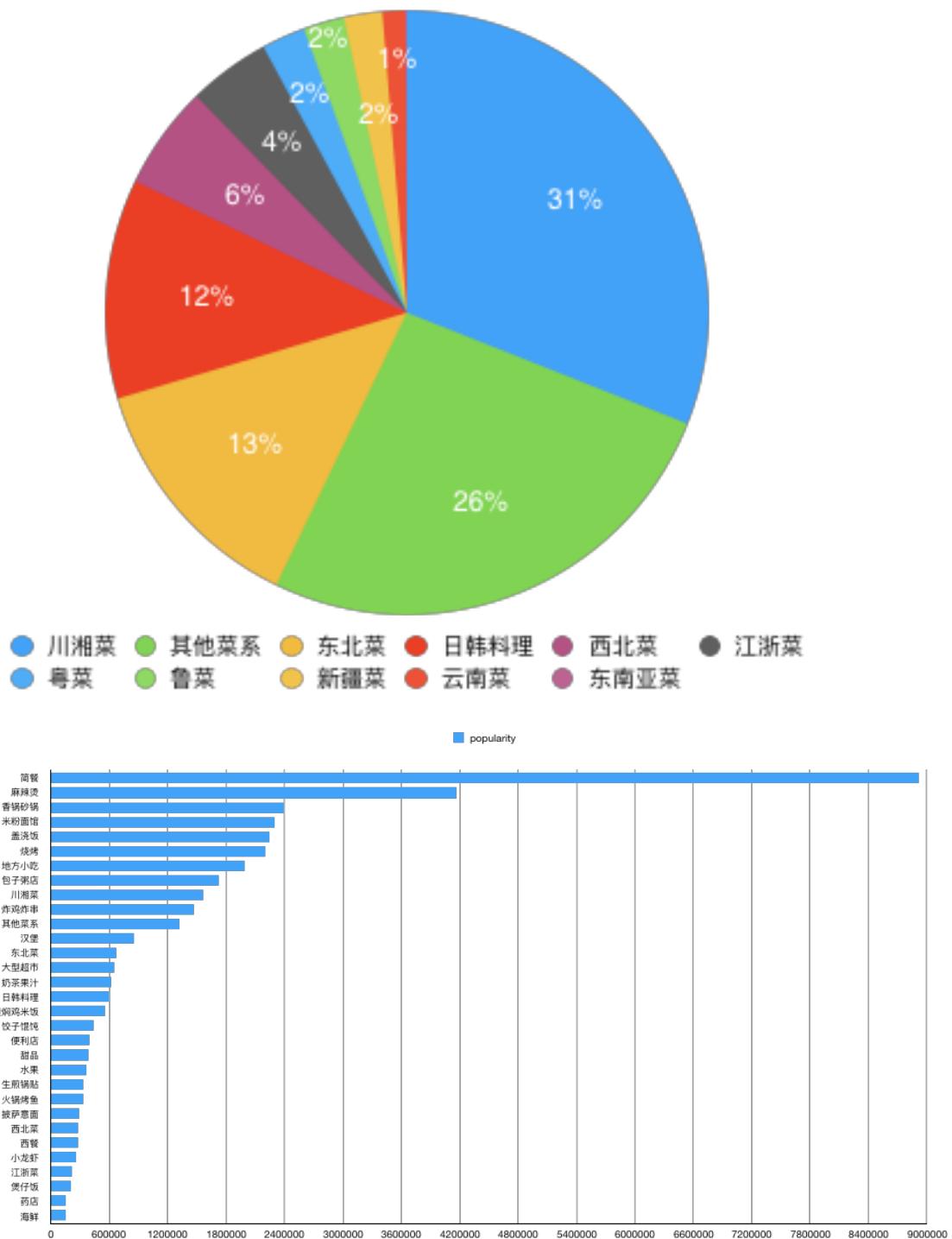
- 目的：分析不同菜系/菜品在不同时段的销量分布
- API: https://www.ele.me/restapi/shopping/v2/menu?restaurant_id=2165785
- 可视化软件：Numbers
- 选取时间：5月31日-- 6月13日（每日9:00am 2:00pm 8:00pm）
- 研究方法：通过对比半个月内不同时段饿了么各餐厅菜单数据，分析菜单中菜品的库存（stock）量变化，研究出菜品销量变化的趋势。



上图分别为销量log化后的变化曲线与一些重要菜式的三维叠层直方图，我们不难发现深夜/晚上的销量总是不及其他时间段，这显然与我们的认知相违背，在



研究学习之前我们认为晚上的销量总体总会较好。而另一方面，烧烤一类的销量总是在深夜/晚上销量较好，这又印证了我们的想法。



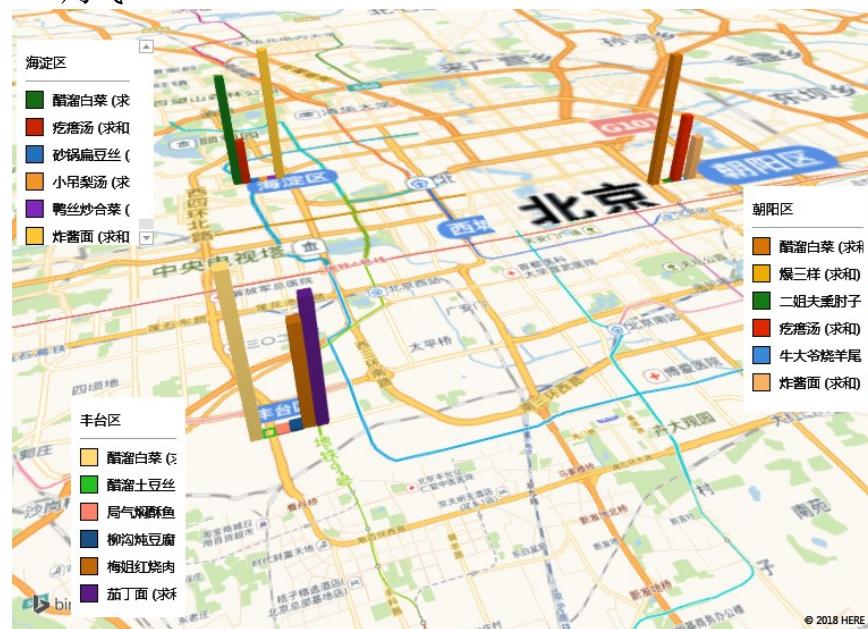
在变化的总销量中，我们可以看到这几本印证了统计平均的总销量占比较大的菜系，如川湘菜的领先地位，但又呈现出局部不同的另一些趋势。比如也许6月份大家更喜欢鲁菜的口味。

第三部分：连锁店分析

- 目标及意义：分析连锁店在北京不同区域内最受欢迎的菜和最不受欢迎的菜，探究区域差异并优化连锁店菜单。
- 划分区域：海淀区、朝阳区、丰台区
- 选择的连锁店：局气（传统餐厅）、必胜客&肯德基（快餐）、鲍师傅（小吃糕点）、coco都可（奶茶）
- 所用可视化软件：Excel & Power map
- 操作流程：从爬下来的数据集中挑出连锁店的菜单及销量数据，比较销量以找出排名前三的菜品和后三的菜品。将统计数据导入power map得到可视化图表

可视化图表：

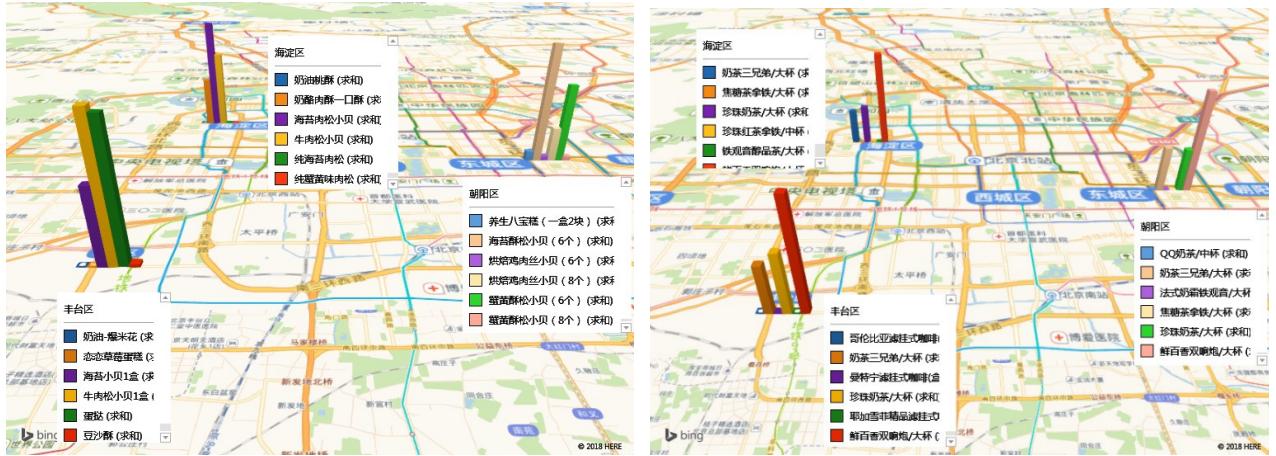
1. 局气



2. 肯德基&必胜客



3、鲍师傅&coco都可



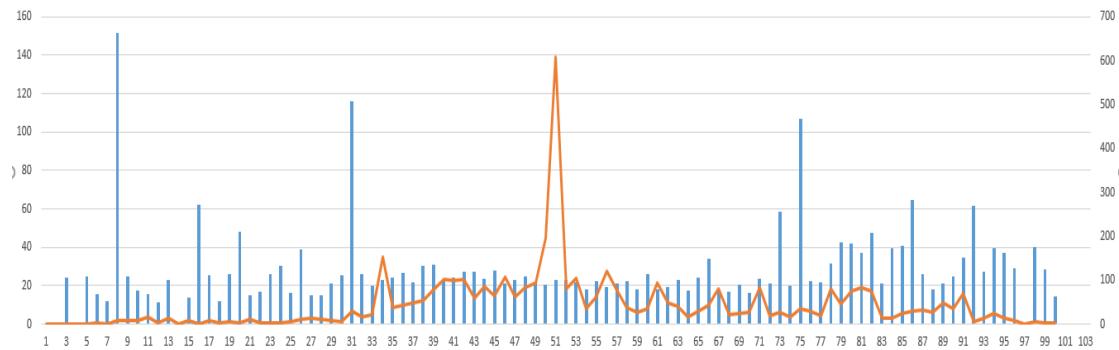
结论

- 不同区域内连锁店最受欢迎的菜基本相同，主要都是某几个菜
- 最不受欢迎的菜存在较大差异
- 原因分析：猜测是由于餐厅菜品比较多，而最受欢迎的菜往往是本店的招牌或某几道菜，处于头部的菜卖的都很火，而其他菜则鲜有人问津。
- 餐馆可以适当的淘汰处于尾部的菜品以优化菜单和餐馆资源配置

第四部分：折扣与销售的相关分析

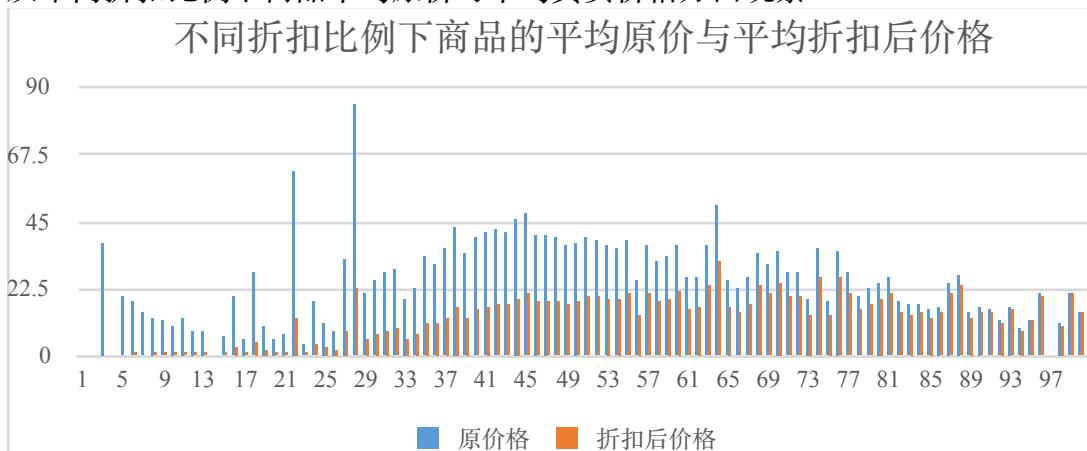
选取数据为：北京地区-折扣商品-售价大于1元-筛选过滤后的5000种商品
横轴为0% - 100% 折扣率

不同折扣比例下的商品种类与平均日销量



参与折扣的商品主要分布在35%-90%折扣率区间（占比92%），其中45%-55%为重点分布区（占比35%）
尽管更多的商家趋向于将商品折扣定选取在35%-70%之间（对客户更有吸引力的折扣区间）
但在这段折扣率区间内商品的平均日售并不突出并且低于70%-90%折扣率下的平均日售

从不同折扣比例下商品平均原价与平均真实价格方面观察



而根据平均折扣价格的变化分类，可大致分为4个区间

0 - 30% 折扣比例区间，折扣比例高，但商品种类参差不齐(多见药品、花卉等特殊外卖商品)，平均折扣价格极低。

30 - 40% 折扣比例区间，存在着随折扣比例提高，商品自身平均价值下降的趋势。

40 - 80% 折扣比例区间，平均商品原价随着折扣比例提高而提高，但平均商品真实价格基本维持不变。即存在更多“先升价后折扣”的商品。

80 - 99% 折扣比例区间的平均原价与40-80区间的平均真实价格接近。而结合折扣率-销量图参考，这部分折扣率商品平均日销量确实高于40-80区间。即80-99区间存在更多的“真实折扣”商品，并且消费者也更青睐这个区间内的商品（在消费者看来也确实是真实折扣）