数据分析报告

Airbnb 超级房东进阶计划

2021 年 05 月

目录

1. 需求分析

1.1 项目背景

近年来，随着互联网技术的飞速发展，以爱彼迎(Airbnb)为代表的一系列民宿、公寓预定平台热度空前。

与传统旅游业的酒店行业不同，Airbnb 由房东自行经营房源，平台仅履行房源发布和统一管理的职责。这样的管理模式不仅赋予了房东极大的自由度，更为旅客提供了个性化、差异化的服务——这也正是其最吸引新时代年轻人的特点所在。根据 Airbnb 官方公布的《中国房东社区报告》，2018 年我国共享住宿的参与人数就已接近 8500 万人；在 2019 年“超赞”房东增长比例达到了 2.6 倍；2020 年更是在全球旅游业面临困境的情况下保持了良好的增长态势。

Airbnb 极高的热度意味着巨大的客户量，无论是站在供给侧的房东还是处于需求侧的旅客，他们对平台频繁的访问操作都在不经意间产生着海量的数据。如果能够对这些数据加以合理的整理与分析，得到可以指导生产实践的结论并运用于实际的平台运作中，无论是平台自身还是使用者都会从中受益匪浅。

基于以上考虑，本项目选择了对 Airbnb 的房源数据进行分析，旨在为平台、房东和感兴趣的投资者提供切实有效的经营管理策略。

1.2 分析目标

在“项目背景”中已经提到，本项目的主要研究对象是 Airbnb 平台上的房源数据，因此所有的准备和分析工作都将从以下三个视角展开。

1. 房源视角

   对于平台和房东来说，投放房源的首要目的毫无疑问是盈利。由于盈利的前提是足够的顾客吸引力，所以如何保证房源获得旅客注意力就显得至关重要。在现有的运营方式中，通常是“经验至上”原则主导着房东对手中房产的管理——例如有阳台的房间可能比没有阳台的房间更容易出租、中式装修风格更受什么年龄段的旅客喜爱等等。

尽管这样的经验通常是行之有效的，但它所需要的时间等各方面成本也是巨大的：手中房产不多或者还很少与其他房东交流的“新手”房东就难以准确把握旅客的喜好，他们可能需要一段时间进行探索，并且这个时间长度通常是难以预测的。

为了解决上述问题，本项目将通过对大量房源本身的各种属性进行详细的分析，从中窥探出“什么样的房源最能够吸引顾客”的规律，为缺少经验的房东和感兴趣的投资者提供有价值的房源运营管理经验。

2. 空间视角

在解决了房源内部装修风格等问题之后，其所具有的一些外部属性同样值得研究和探讨。以房源地理位置为例，究竟是靠近商圈的房源还是地处城郊的房源更能打动旅客呢？或者说：不同的城市是否会对促使旅客对房源的外在条件提出不同的要求呢？这些非常具有现实意义的问题由于数据量大、格式杂乱等问题使大多数研究者望而却步。

秉持为平台和房东提供一站式解决方法的想法，本项目将会选择多个城市的 Airbnb 房源信息进行统计和对比分析，帮助平台、房东及相关投资者对不同的房源采取具有针对性的宣传和个性化的管理方法。

3. 房东视角

在现实生活中，旅客在选择民宿时通常不会仅仅关注于房源本身的“硬性条件”，房东自身的一些行为属性对游客的决定也扮演着至关重要的角色——房东的信息完善度会对旅客的决定产生什么样的影响？房东回复评论的情况是不是也在暗中影响着旅客的心意？上述问题只是从房东视角出发可以探索的问题的冰山一角。

不难看到这样的一个切入角度是非常具有价值的，但是现在却鲜有相关的研究和结论。原因也很显然：巨大的房东群体很难集中在一起进行人为的调研和分析。因此，本项目将通过机器学习对以上探索过程进行自动化，用最小人力成本探寻隐藏在背后具有现实意义的“好评房东行为指南”。

2. 数据获取

2.1 数据来源

为了使项目的结论真实可靠且保证研究过程符合相关法律法规，本项目选择了由 Airbnb 官方提供的北京、上海两座城市在 2012 至 2021 年的房源信息统计数据。根据官方提供的数据使用条例：所有的数据均已进行脱敏处理，避免了房东和旅客隐私泄露的问题，进一步确保了项目开展的真实性与合法性。

2.2 获取方式

从官网获取数据的流程并不复杂，共分为以下步骤，均可通过浏览器完成。

1) 登录 Airbnb 数据平台：http://insideairbnb.com/get-the-data.html
2) 选择城市：本项目选择北京和上海两座城市
3) 在 File Name 栏点击超链接下载到本地解压即可

3. 数据清洗

3.1 原始数据

以北京市房源为例（上海市房源分析流程及结论相同），从官网下载的数据解压后可以得到以下文件：

a) listings.csv：包含所有的房源和房东信息。该数据集存在着很多格式结构上的问题：房东和房源的信息混杂在一起难以区分、域中的值不符合 csv 文件规范等等。值得一提的是：官网提供了一个简略的 listings.csv 文件供研究者使用，其数据、格式相对而言更为规范，但是也存在属性数量、整体样本较少等难以从根本上解决的问题。因此，本项目选择了 Airbnb 平台公布的原始数据作为基础进行分析。
b) reviews.csv：包含所有房源的评论信息

尽管官方数据非常的全面，但是由于庞大的数据量以及粗糙的预处理操作，

其完整性受到了极大的破坏，很多的信息已经没有办法继续使用了。以房源相关属性为例，有 33.4%的房源没有对应的行政区划名；所有的房源都没有浴室数量统计；甚至评分项存在接近一半(44.7%)的缺失。从以上的分析不难得到以下结论：如果直接使用原始数据进行分析，很明显是不能够得到具有现实意义的规律和真正有效的管理策略的。通过对属性进行分类和 EDA（Exploratory Data Analysis，试探性数据分析）后，可以统计整理得到如下的概况信息。

表 3.1 房源属性

属性名	说明	属性名	说明
room_id	房源编号	bathrooms	浴室数量
host_id	房东编号	beds	床铺数量
neighbourhood	行政区划	price	价格
name	房源名	amenities	设施
latitude	纬度	instant_bookable	可随时预定
longitude	经度	first_review	首次评论时间
room_type	房源类型	last_review	最近评论时间
accommodates	可容纳人数	number_of_reviews	总评论数
bedrooms	卧室数量	review_scores_rating	评论分数

表 3.2 房东属性

属性名	说明	属性名	说明
host_id	房东编号	host_has_profile_pic	是否有房东照片
host_name	房东名字	host_identity_verified	是否信息认证
host_since	房东入驻时间	host_response_rate	评论回复率

表 3.3 原始房源数据信息统计（单位：个）

城市 / 数据集属性	北京	上海
房源总量	24977	36294

总缺失项	69071	97733
某项缺失率	15.4%	15.0%
重复项	0	0
某项重复率	0.0%	0.0%
数量属性	10	10
类别属性	7	7
布尔属性	1	1

表 3.4 原始房东数据信息统计（单位：个）

城市 数据集属性	北京	上海
房东总数	24977	36294
总缺失项	5770	6664
某项缺失率	3.9%	3.1%
重复项	15470	26103
某项重复率	61.9%	71.9%
数量属性	1	1
类别属性	3	3
布尔属性	2	2

3.2 清洗原则

为了能够充分利用原始数据具有的数量大、信息全面的优势，同时尽量避免其中的无效数据或错误数据为分析带来负面影响，正式分析前的数据清洗就显得尤为重要。在数据清洗时，需要严格遵守以下原则，避免影响原始数据的可靠性。

- **保留有效数据，对无效、错误数据进行过滤。**例如已经下架却仍出现在数据集中的房源需要进行删除操作。
- **不填充空值，保证数据的真实性。**例如某些房源可能存在评论时间不存在的

情况，在清洗阶段不对这一部分数据进行处理，等到清洗完成后对有效的、干净的数据采取相关的填充操作。

- **数据规范化，保证结构统一。**例如某些房源的行政区划为空或过于模糊（仅指明了“北京市”等），由于所有房源的经纬度均有效，可以对该域进行操作使所有条目均细化到相同粒度。

3.3 清洗方法

由于数据量较大，采用人工的方法显然不可取。本项目使用了 Python 对与原始数据进行了数据清洗和整理，具体步骤如下，代码实现见附录 A 部分。

A. 房源信息

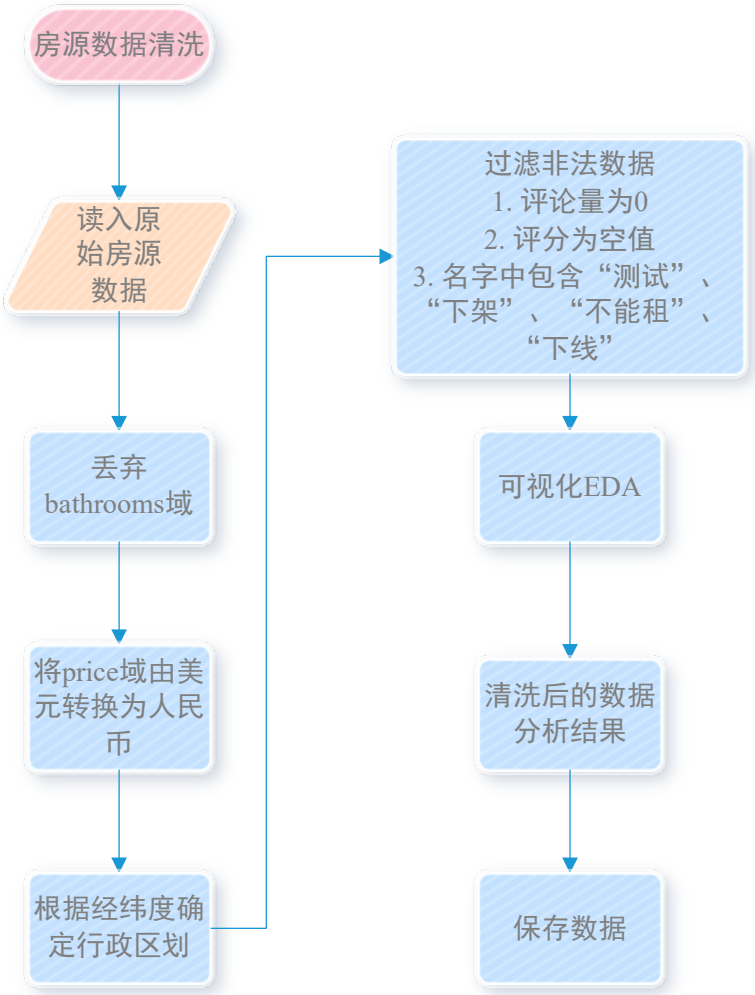

图 3.1 房源数据清洗流程

B. 房东信息

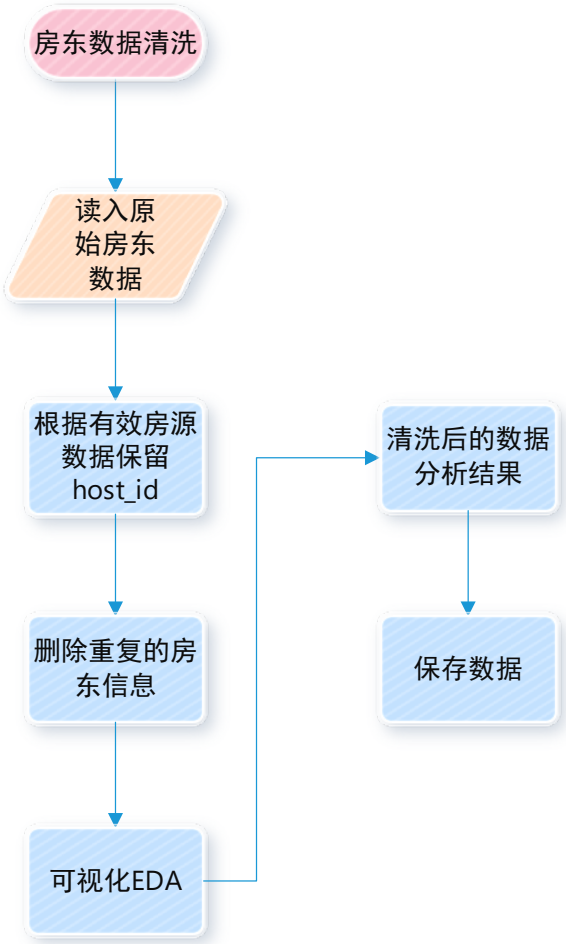

图 3.2 房东数据清洗流程

3.4 清洗结果

在按照上述流程对数据完成清理后，对新数据集进行分析得到结果如下。

3.4.1 数据集文件

1) room.csv：包含了所有有效的房源信息。

表 3.5 清洗后房源数据信息统计（单位：个）

城市 / 数据集属性	北京	上海
房源总量	9438	14875
总缺失项	161	263

某项缺失率	0.1%	0.1%
重复项	0	0
某项重复率	0.0%	0.0%
数量属性	10	10
类别属性	6	6
布尔属性	1	1

2) host.csv：包含了所有有效的房东信息。

表 3.6 清洗后房东数据信息统计（单位：个）

城市 数据集属性	北京	上海
房东总数	4724	4865
总缺失项	1044	890
某项缺失率	3.7%	3.0%
重复项	0	0
某项重复率	0.0%	0.0%
数量属性	1	1
类别属性	3	3
布尔属性	2	2

可以看到，在按照上述流程对数据进行清洗后，缺失率有了显著的下降。这说明原始数据中的确存在大量没有意义的无效数据，将它们纳入考虑不仅增大了模型负担，还可能对结论造成影响，进一步证明了数据清洗的必要性和有效性。

3.4.2 可视化 EDA

在清洗之后，由于缺失值减少和格式的进一步完善，可以进行原始数据难以支持的可视化分析。以北京市的房源信息为例，在通过经纬度将行政区划的粒度统一后，通过简单的计数和绘图即可得到民宿在各个区的分布情况如下。

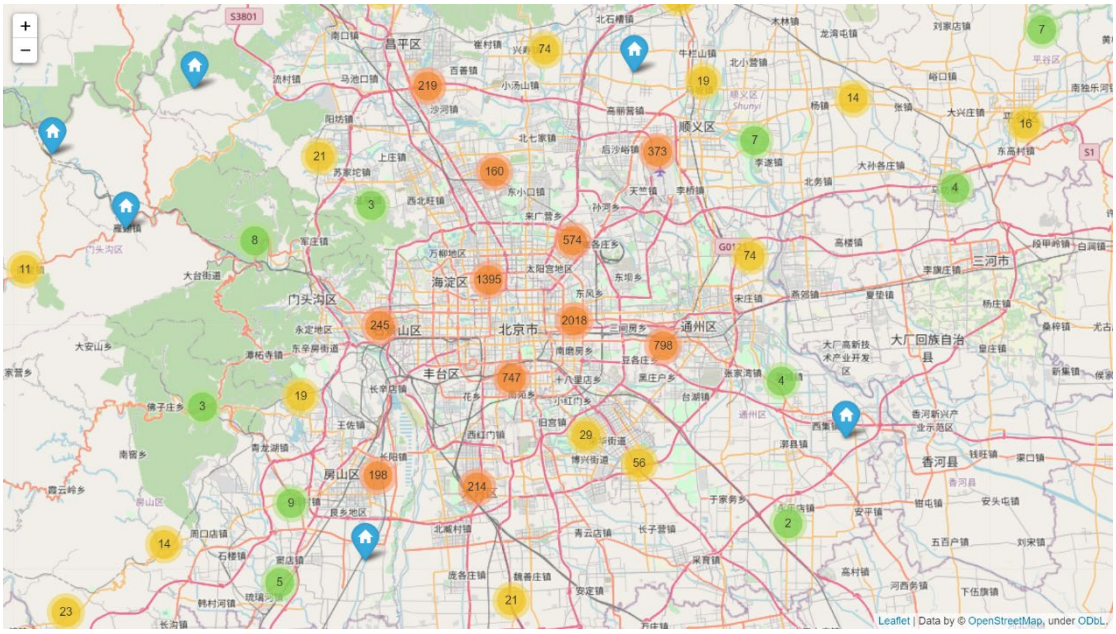

图 3.3 北京市民宿分布情况

Value	Count	Frequency (%)
朝阳区	3184	33.7%
海淀区	883	9.4%
丰台区	806	8.5%
密云区	688	7.3%
延庆区	565	6.0%
怀柔区	560	5.9%
昌平区	486	5.1%
通州区	457	4.8%
顺义区	456	4.8%
大兴区	383	4.1%
Other values (6)	970	10.3%

图 3.4 北京市民宿分布占比

在数据清洗之前，这些操作都是不可能完成的，更不用说直观地得出“朝阳区拥有北京最多的民宿且远超第二名”的结论了。这样定性与定量的双重认识是在数据清洗后才可能获得的，足以说明数据清洗对接下来分析过程的重要性。

4. 数据整理

前期数据获取与清洗工作保证了数据的有效性和可靠性，为进一步的数据分析工作打下了良好的基础，但这并不意味着可以马上对数据开展分析操作了——现在的数据集还并没有将“语义”信息显露出来。换句话说，现在的数据是正确的，但是业务逻辑是不够明确和规范的。以房源数据中的租金属性为例，依据数据清洗准则可知：所有的租金记录都是有效的，但是极差也是巨大的。动辄几十万的租金作为数据中的离群点尽管出现频度较低，但由于其偏离程度过高，也有可能对最终的分析结果造成巨大的影响。

因此，为了得到真实可靠且具有实用价值的普适结论，本项目在正式分析数据之前对数据进行了数据预处理、特征工程等数据整理工作，具体步骤将在本节中详细阐述，代码实现见附录中的 B、C 部分。

4.1 数据预处理

A. 房源信息

1. **拆分“amenities”列单独分析。**该列表示房源内部有哪些基本设施和工具，例如 WiFi、防火栓、吹风机等。由于各个房源的 amenities 列属性较多，因此该列需要单独进行分析，故根据基本设施拥有情况进行拆分。
2. **删除租金为 0 的房源。**极少数房东入驻 Airbnb 并不是为了盈利，因此不对旅客收取房租。但这与本项目的需求相悖，故删去。
3. **根据 3σ 原则对租金属性进行筛选。**对各个行政区划而言，均有可能会出现 10 万元以上的“特高租金”。而实际情况中这些房源占比极小，却因为极大的偏差对平均价格等分析依据造成了不可忽略的影响。因此，为了贴合本项目面向的大多数房东和投资者，对这些离群点选择删除操作。
4. **使用均值对各个属性的缺失值进行填充。**得益于前期的数据清洗工作，现阶段房源数据集中的数据缺项已经降至了 0.1%.因此，直接使用均值对缺失值进行填充可以最大程度上保留原始数据集中蕴含的信息。
5. **对部分属性的数据类型进行规范。**原始数据中对百分比的表示可能是字

符串形式的 90.5%，但是计算机无法正确识别。因此，将此类域统一为计算机可以正确识别和处理的表示形式。

B. 房东信息

1. **使用均值对各个属性的缺失值进行填充。**具体原因和房源信息数据预处理中第 4 步相同。
2. **对部分属性的数据类型进行规范。**具体原因和房源信息数据预处理中第 5 步相同。

4.2 特征提取与构造

A. 房源信息

1. **筛选“优质房源”数据。**设置优质房源评判标准：评论数大于 90%房源且评分高于 90%房源。根据该标准在房源数据集中进行选择，得到所有的优质房源信息并进行存储。

B. 房东信息

1. **统计房源拥有量。**根据房东和房源的共同属性 host_id 对房东名下的房源数量进行统计。
2. **确认房东入驻年份。**根据房东 host_since 属性进行解析，从中提取出房东入驻 Airbnb 的年份。
3. **计算房东总收入。**将房源的评论数作为房源的有效出租次数，同时结合房东和房源的对应关系来计算得到房东入驻平台后的总收入。
4. **计算房东日平均收入。**根据房东总收入和入驻时间的商求出日平均收入。
5. **计算房源日平均收入。**在房东日平均收入的基础上，再考虑其所拥有的房源数量，通过计算得到房源日平均收入，及每套房源每天能为房东提供多少收益。
6. **筛选“超级房东”。**根据优质房源的统计信息对房东进行标记——凡是拥有优质房源的房东都是超级房东。
7. **对房东进行分类。**对于拥有 5 套及以上房源的房东，打上“企业房东”标签，其余为“个人房东”。

5. 描述分析

在完成了前期数据获取、数据清洗、数据整理的工作后，现有数据的格式、业务逻辑信息已经比较完善和充分了。接下来，本项目将正式对 Airbnb 平台使用过程中产生的房源、房东数据进行分析，紧贴业务逻辑，旨在为平台、房东和感兴趣的投资者提供切实有效的经营管理策略。

5.1 行业现状——Airbnb 是否仍值得进场？

5.1.1 结论

在 2021 年，Airbnb 仍然是一个值得房东和投资者进行房源投放的租赁平台。

5.1.2 分析证明

1. Airbnb 市场规模巨大，且存在巨大发展潜力。

从下图中可以看到，Airbnb 房源具有分布广、数量大的特点，与传统租房行业基本一致。更重要的是，传统租房行业并没有集中的统一管理机构，这意味着传统租房行业难以在规范性和宣传力度等层面与 Airbnb 相媲美。

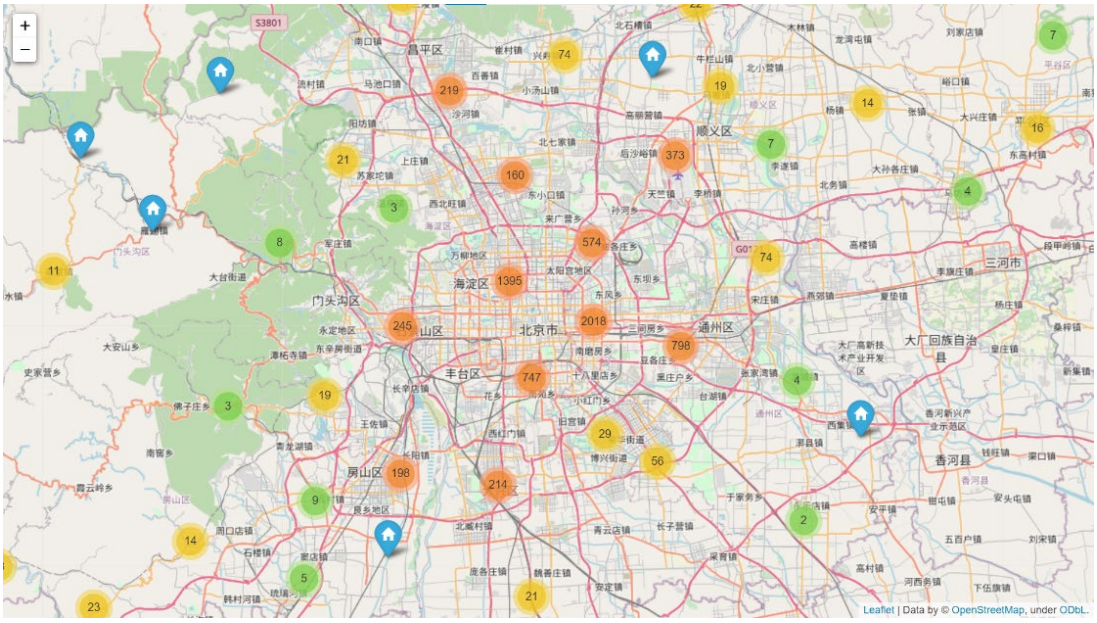

图 5.1 北京市 Airbnb 房源地理位置分布

进一步地，从北京市 Airbnb 房源数量的统计情况可以看出：现有市场还存在较大的上升空间。以朝阳区为例，其面积为 470.8 平方公里，约占北京市非山区面积的 7.43%.但该区却因为 Airbnb 发展较早而拥有约 33.9%房源占比。因此，整个北京市区内还有很多可能的民宿没有被开发，市场潜力依旧可观。

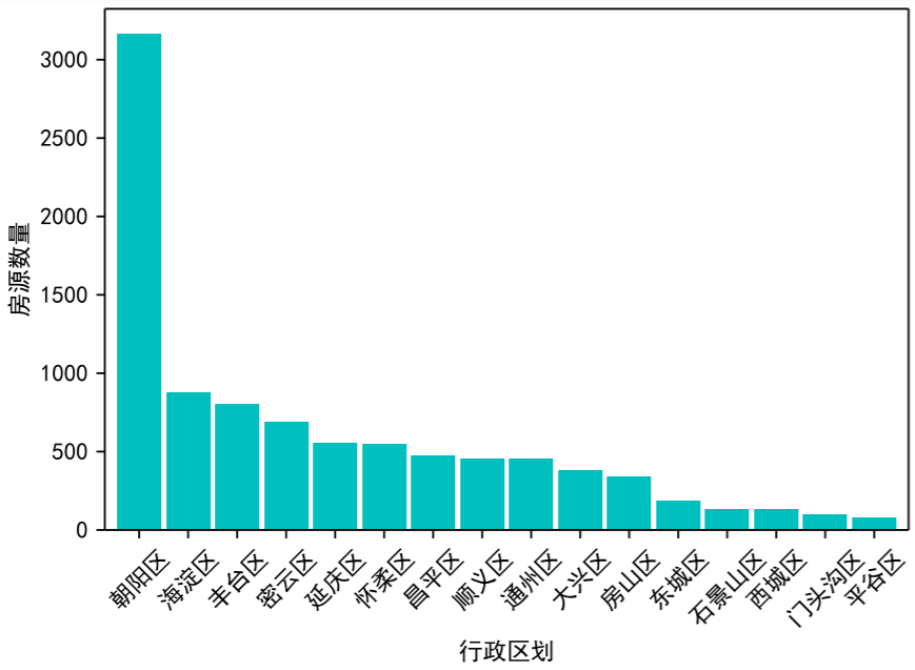

图 5.2 北京市 Airbnb 房源数量分布

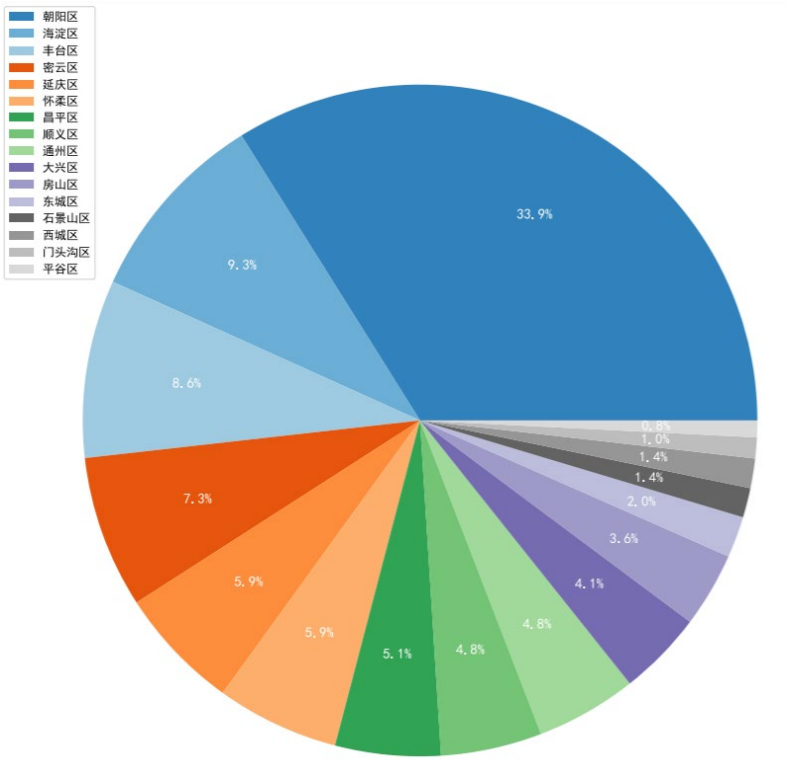

图 5.3 北京市 Airbnb 房源数量占比

2. Airbnb 房源种类多样，绝大部分房源均可以找到对应受众。

从绝对数量上，Airbnb 虽无法与传统租房行业相提并论，但是在广度上 Airbnb 并不逊色半分。以房源价格为例，在经过数据整理阶段的降噪操作后，现有记录中已经不再存在日租金 10 万甚至 40 万以上的极端案例了。

但是，从统计情况中不难发现：房源租金的分布依旧十分广泛。以房源数量最少的平谷区为例，其最低租金在 500 元左右，而最高租金却达到了 40000 元。同时结合图 5.5 中的价格分布情况可以证明：各个行政区中均可以支持从简约到豪华的多种类型房源运营。而对于传统租房行业来说，由于缺少集中的管理和统计，很难保证各类型的房源均能够创造对应的最大收益。

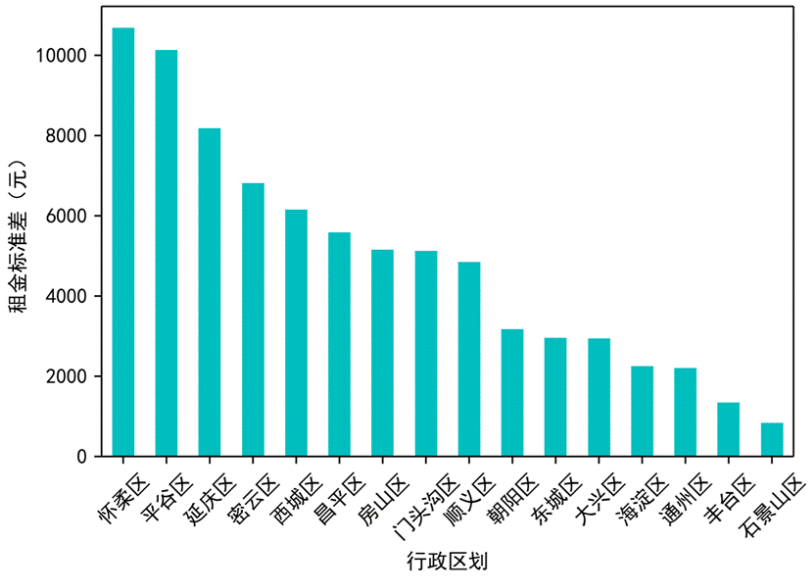

图 5.4(a) 北京市 Airbnb 房源租金标准差统计

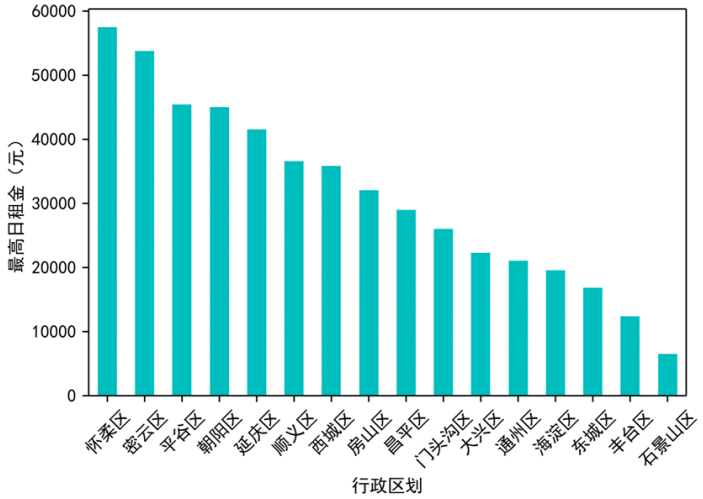

图 5.4(b) 北京市 Airbnb 房源最高日租金

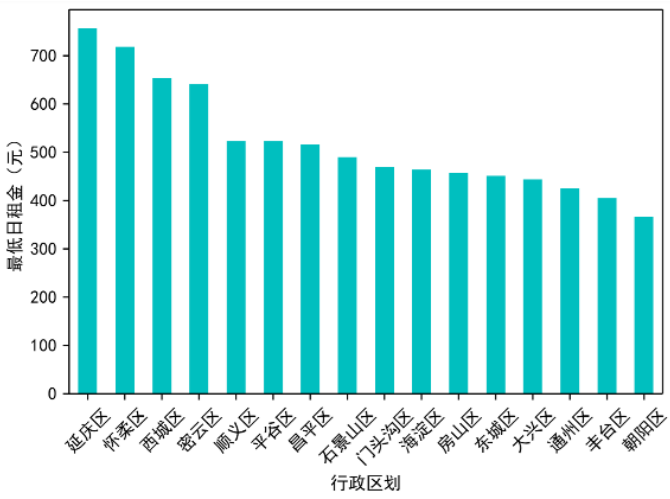

图 5.4(c) 北京市 Airbnb 房源最低日租金

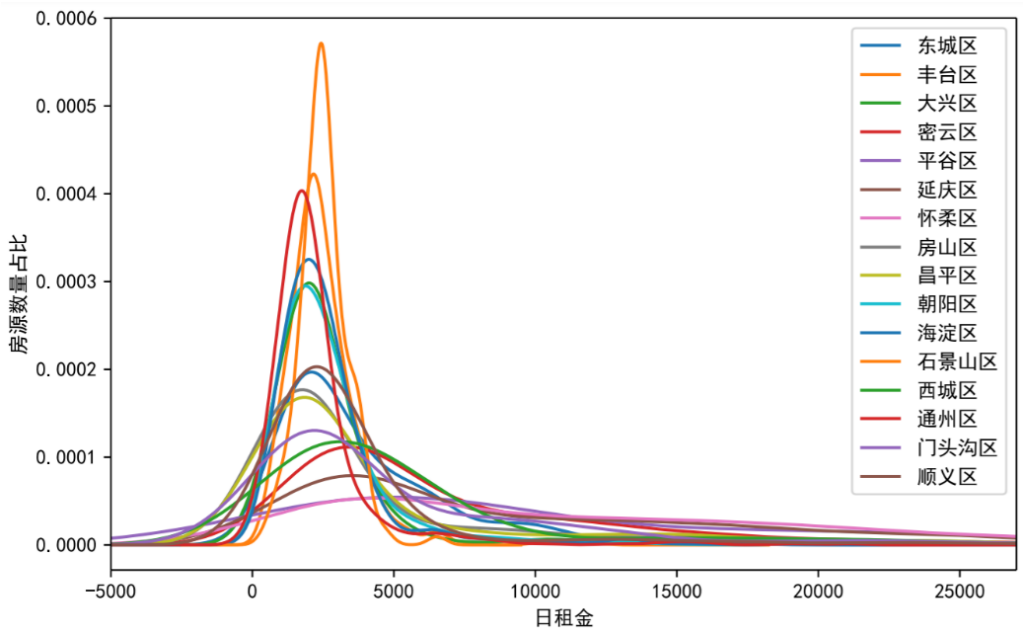

图 5.5 北京市 Airbnb 房源租金分布

3. Airbnb 房东数量和租金收入依然在稳步增长。

自 2011 年 Airbnb 进入北京市场开始，入驻的房东数量每年都在稳步增长。尽管 2020 年受到疫情影响全球酒店行业都面临着巨大的生存挑战，Airbnb 平台依然实现了房东数量的正增长。更值得注意的是，这十年间新入驻的房东其房源日平均收入也同样保持着良好的增长态势，特别是 2020 年入驻的新房东更是打破了 2016 年创下的最高收入记录。疫情时期的逆风翻盘，足以说明 Airbnb 能够为房东提供良好的发展机遇和前景。

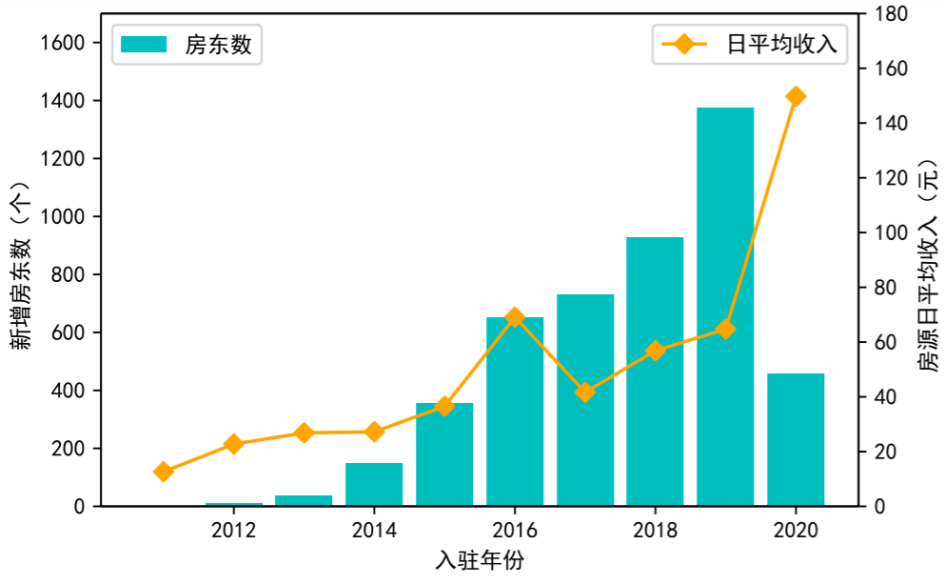

图 5.6 北京市 Airbnb 房东入驻及收入情况统计

4. Airbnb 个人房东和企业房东均能够发挥优势获取收益。

从房东构成情况中可以看到：Airbnb 平台上超过 90%的房东都是自主经营，且两种房东在收入上各有千秋。对于个人房东而言，主打服务质量。因此从单个房源的平均收入来说，个人房东是能够获得更多的收益的。

同样的，企业房东利用其房源多、统一管理的优势，能够在总收入上超越个人房东。因此，无论是个人房东还是企业房东均能在 Airbnb 平台占有一席之地。

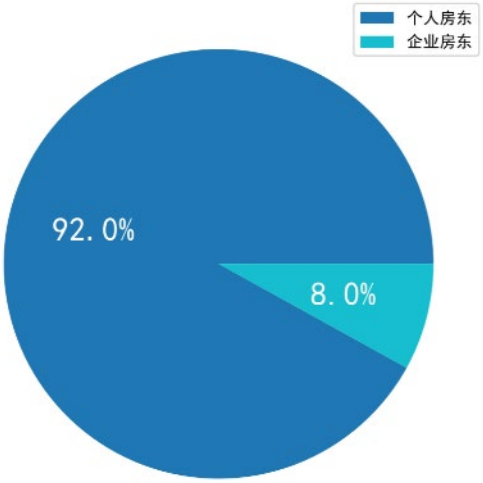

图 5.7 北京市 Airbnb 房东类型占比

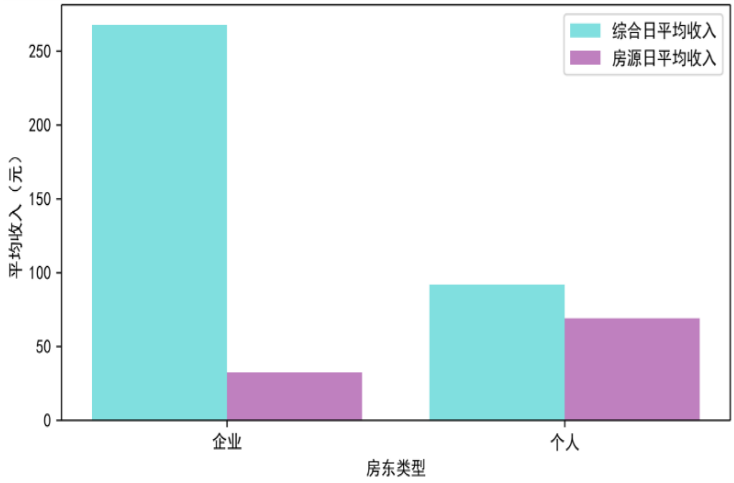

图 5.8 北京市 Airbnb 房东类型及收入情况统计

5. Airbnb 超级房东收入高于传统整套月租房东。

根据数据整理阶段的工作，Airbnb 房东被划分为了超级房东和普通房东。为了进一步说明 Airbnb 的市场红利依旧可观，项目将北京 Airbnb 房东与整套月租的传统房东进行了收入比较。可以看到，由于整套月租能够保证固定收益的特殊性质，其平均收益肯定是高于日租的 Airbnb 房源的。

但是如果 Airbnb 房东能够认真经营升级为超级房东，那么其日收益会超过传统房东。同时，Airbnb 房东还能够更灵活地管理自己的房源，例如出租几天后自己再使用，这是传统方式不能保证的。因此，在比传统方式更加规范、安全、可靠的环境中出租自己的房源，结交更多有趣的朋友，还能够得到更高的收益，Airbnb 的吸引力可见一斑。

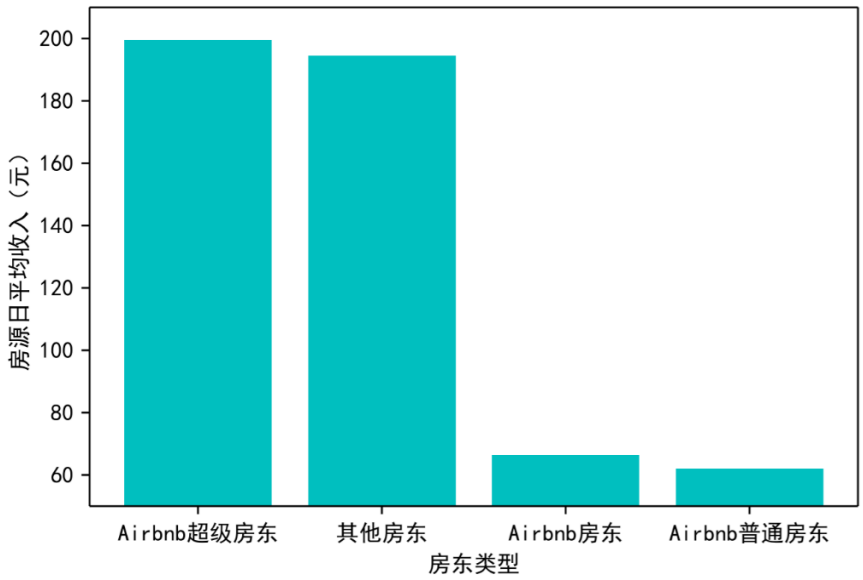

图 5.9 北京市各类型房东收入比较

5.2 房源解析——什么样的民宿招人爱？

5.2.1 结论

1. 房源的地理位置很重要，应优先选择朝阳区、丰台区、海淀区。同时，靠近地铁（特别是 10 号线）、国贸、三里屯、望京、798 艺术中心、天坛和王府井等景点的房源更能创收。
2. 在条件允许的情况下，非大型房源（可容纳 10 人以上）应尽量拆分为多个小型房源（可容纳 5 人及以下）进行出租，且最好是单间。
3. 房源的一般属性，如是否可以即时预订等，对房源盈利能力影响不大。

5.2.2 分析证明

1. 民宿价格与多个因素有关，且含明显的线性相关性。

对于房东来说，最关心的毫无疑问是房源的价格，也即自己手中的房源能够创造多大的收益。因此，本项目对房源的基本属性进行了相关性分析，得到相关性矩阵如下。

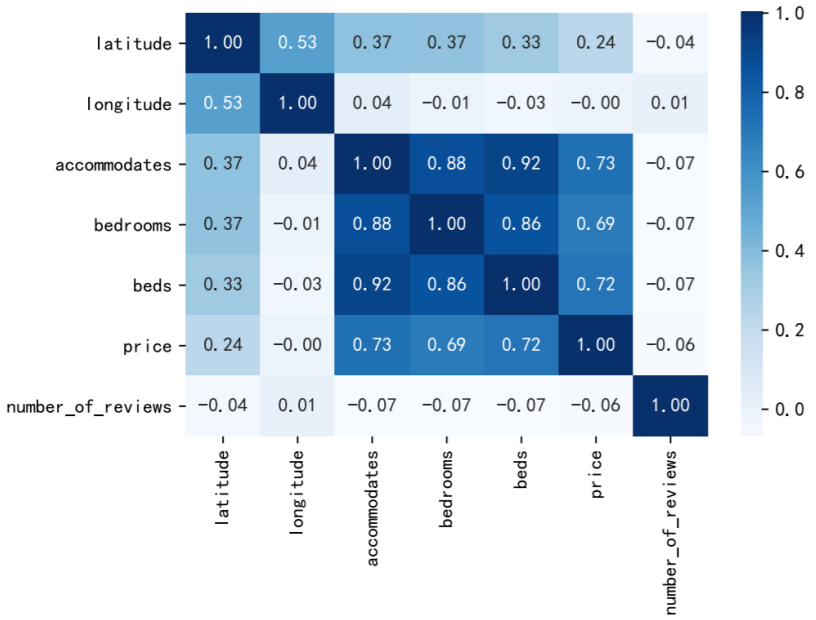

图 5.10 北京市 Airbnb 房源基本属性相关性矩阵

从图中可以看到，房源的价格和维度、容量、卧室数量等均具有比较明显的相关关系。例如从图 5.11(a)中的散点分布可以看出：房源容量和价格基本是线性正相关的。出人意料的是，房源的评论数量和最终的价格并没有明显的相关性。价格高自然受众会相对较少，评论数也会相应较少。但是在价格适中的房源中，也并不一定会有比价格高的房源更多的评论数。这说明房源价格对旅客是否下单并没有明显影响，再一次说明了 Airbnb 房源种类多样，且均能够找到对应受众的事实。

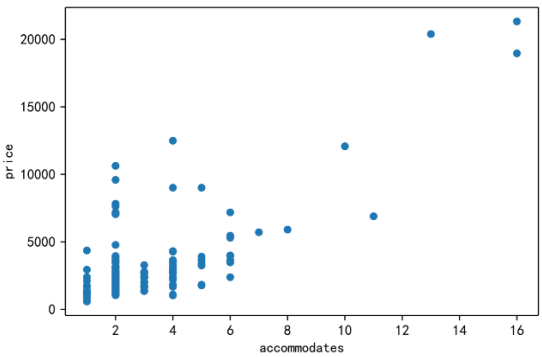

图 5.11(a) 北京市 Airbnb 房源容量与租金分布散点图

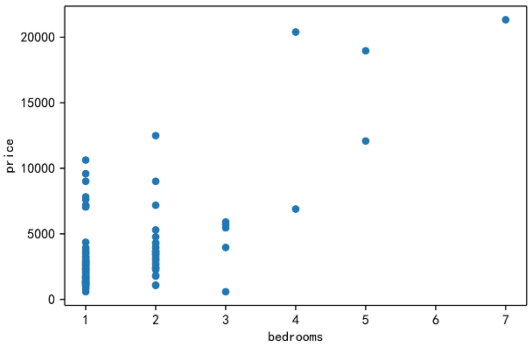

图 5.11(b) 北京市 Airbnb 房源卧室数量与租金分布散点图

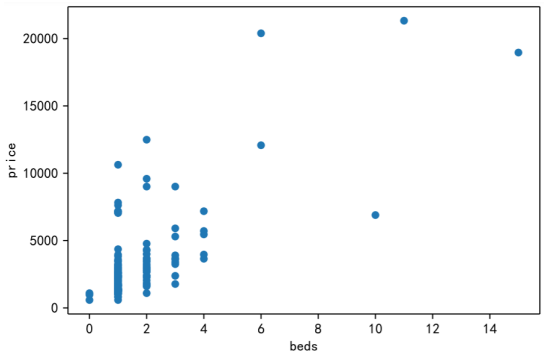

图 5.11(c) 北京市 Airbnb 房源床数量与租金分布散点图

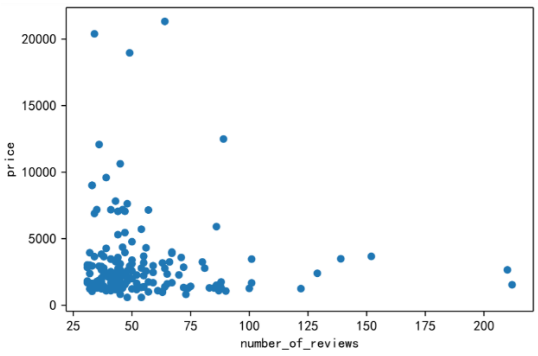

图 5.11(d) 北京市 Airbnb 房源评论数量与租金分布散点图

2. 位于朝阳区、丰台区、海淀区的民宿更受欢迎。

承接数据整理阶段对优质房源的筛选工作，本项目进一步对优质房源在各个行政区的分布数量进行了统计，可以看到前五分别是朝阳区、丰台区、海淀区、顺义区和密云区，这基本符合常识。

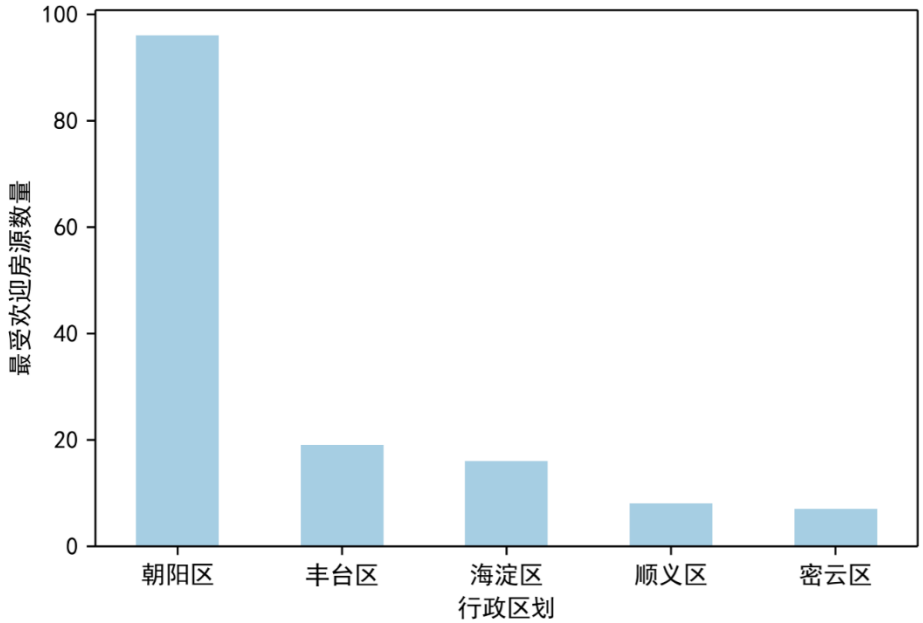

图 5.12 北京市最受欢迎 Airbnb 房源数量前五行政区划

但是否这就足以说明这几个区是优质房源的大本营呢？答案是：目前简单的排序工作还不足够证明以上论断。

这里先对“幸存者偏差”进行简要的介绍：这个理论告诫数据分析工作者，在分析现象时不能仅仅对某一类样本进行采样，否则得到的规律是不具有现实意义和普适性的。具体来说，对于行政区划产出优质房源的能力，如果仅仅考虑优

质房源的分布而忽略了这些全体房源在这些行政区的分布，那么得到的结论是不可靠的。因此不能仅仅考虑优质房源在各个区分布的绝对数量，更重要的应该是某个行政区划对优质房源的转化系数，通过以下公式进行计算。

$$优质房源转化系数 = \frac{优质房源占比}{房源占比} \tag{1}$$

接下来对公式进行展开：首先是转化系数的分子“优质房源占比”，该变量表示某个行政区所拥有的优质房源占总优质房源数量的比例。体现了对优质房源绝对数量的考虑，即优质房源数量越大的行政区划该变量就会越大，通过下式进行计算。

$$优质房源占比 = \frac{某行政区优质房源数量}{优质房源总数量} * 100\% \tag{2}$$

然后是公式中的分母“房源占比”，该变量表示某个行政区所拥有的房源占全体房源数量的比例。体现了房源绝对数量的考虑，即房源数量越大的行政区划该变量就会越大，通过下式进行计算。

$$房源占比 = \frac{某行政区房源数量}{房源总数量} * 100\% \tag{3}$$

公式解析到这里，其逻辑已经十分清楚了：朴素的想法仅考虑了分子“优质房源占比”，这是经不起推敲的。假设存在某行政区划 A，其占有了整座城市 90%的房源和 80%的优质房源；对应的 B 区分别占有 10%和 20%。如果按照朴素的想法，那么 A 区就是最适合房东进行房源投放和投资的区域。但事实上，A 区对优质房源的转换系数连 1 都不到，而 B 区的优质房源转化系数却是 2.因此，B 区才是堂堂正正的“英雄”。

按照以上思路，所有转化系数大于 1 的行政区划才是真正具有较大房源投资价值的选择。从下图中可以看到，原本前五的朝阳区、丰台区、海淀区、顺义区和密云区中就出现了上述 A 区的情况。作为前三的朝阳区、丰台区、海淀区守住了擂台，的确是北京市最能够产出优质房源的风水宝地。但是顺义区却下滑到了第 5 名，且其转化系数是小于 1 的，也就是说它占了房源基数大的便宜。更有甚者，密云区的转化系数并未位于前 5 之列，它所享受的房源基数“优势”在此尽

数体现了。综上所述，最受欢迎的房源主要出现在朝阳区、丰台区和海淀区，它们也是最适合房东投放房源的三个行政区。

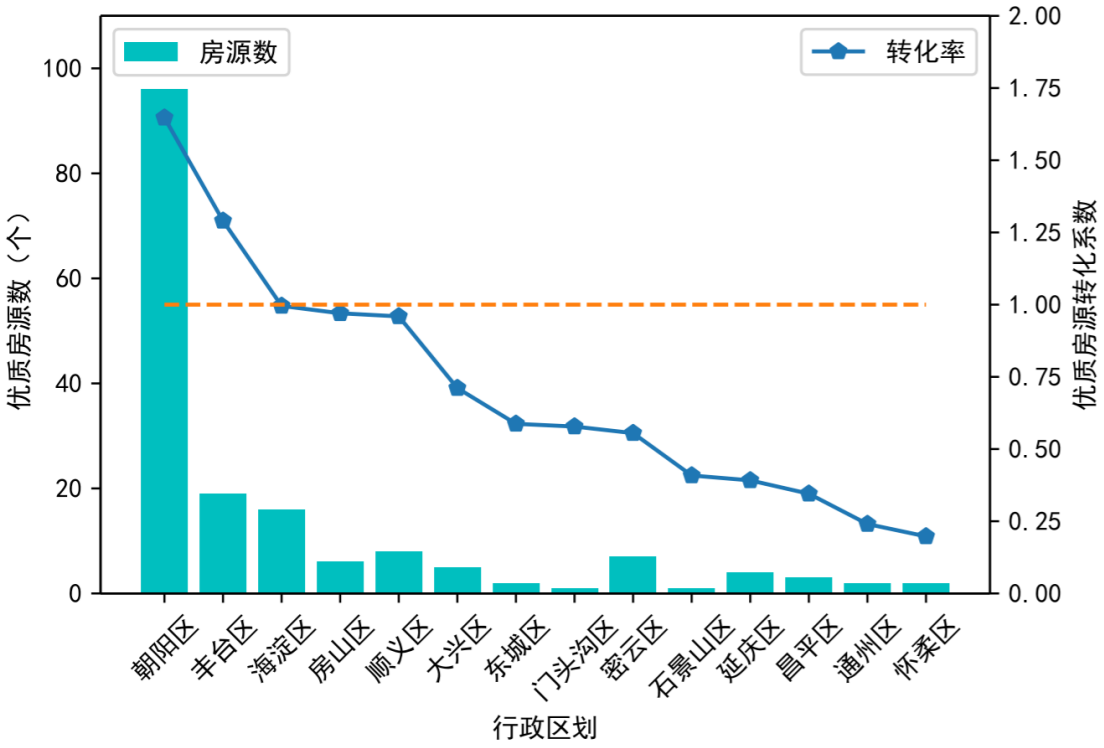

图 5.13 北京市 Airbnb 优质房源行政区划转化情况

3. 相比于中型房源，小型房源和大型房源更受青睐。

对于房源容量来说，同样应该采用“转化系数”作为第一考虑要素。从下图中可以看到，5 人及以下的小型房源和 11 人及以上的大型房源更容易成为优质房源。可以作如下解释：使用 Airbnb 的旅客通常是以个人或小家庭为单位，因此小型房源不仅转化系数高，所占比例也同样很大。对于大型房源而言，通常承接的是聚会、派对等活动，相对而言样本数量较小，但这些大型房源的经营者通常都有丰富的管理运营经验，因此更能够获得客户的喜爱。而介于 5 和 11 人之间的中型房源可能由于受众并不清晰而鲜有优质房源脱颖而出。

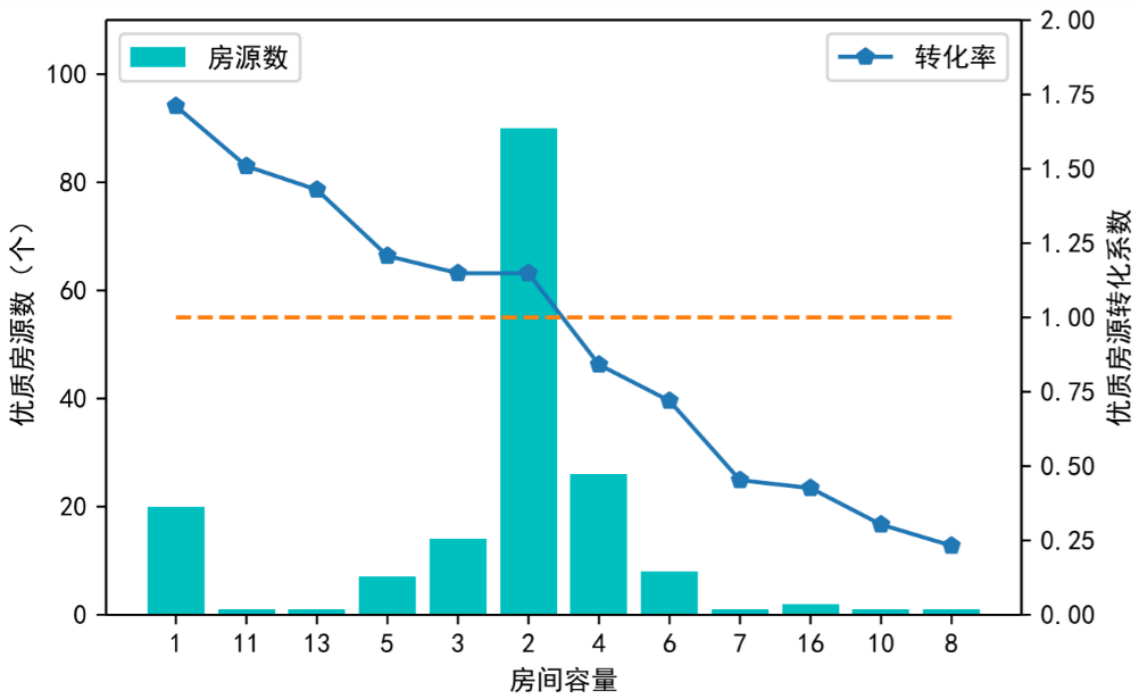

图 5.14 北京市 Airbnb 优质房源容量转化情况

4. 单间是所有民宿类型中最受住客喜欢的房源类型。

在房源的三种类型中，单间最能够迎合 Airbnb 主要的客户群体：个人。同时，由于单间的维护、清洁、入住等流程都十分简单便捷，对房东和房客来说都是非常具有吸引力的选择。因此，单间是最值得房东进行运营的房型。

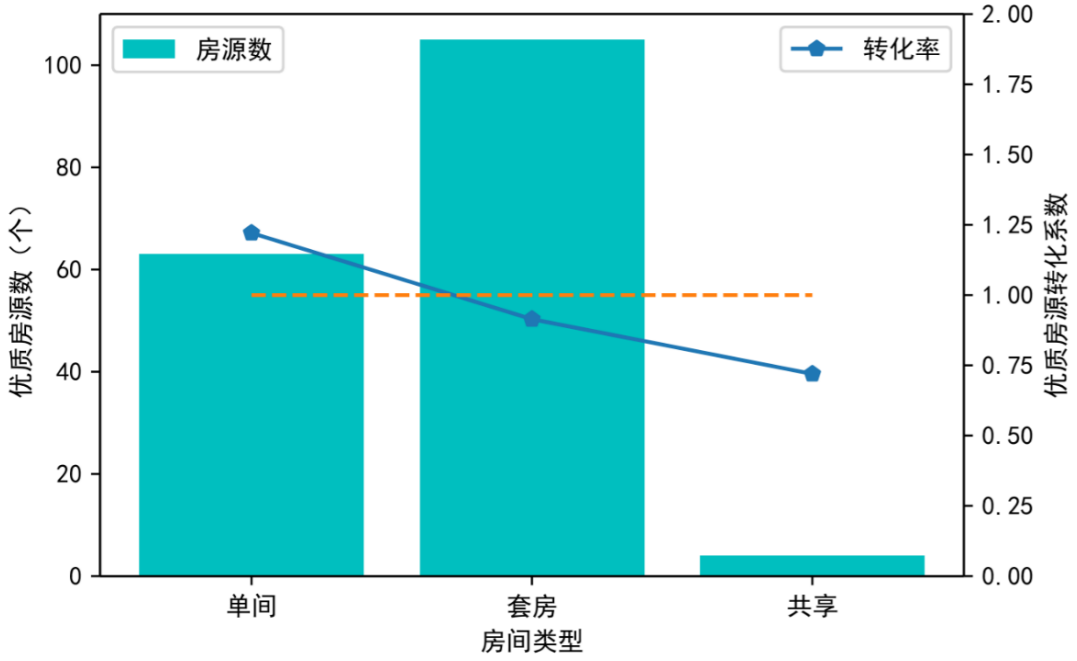

图 5.15 北京市 Airbnb 优质房源类型转化情况

5. 靠近地铁（特别是 10 号线）、国贸、三里屯、望京等景点的公寓更容易获得旅客的好评。

在处理房源描述中的关键词时，由于其种类繁多且差异巨大，因此不能够直接沿用上述“转化系数”思想来规避“幸存者偏差”。本项目采用了对分词结果进行频度校正的操作。具体来说，在优质房源和普通房源数据中均进行关键词统计并进行排序，然后利用普通房源的关键词频统计对优质房源的关键词进行频度校正。

围绕“寻找优质房源有而普通房源没有的特点”这一中心思想，对于在优质房源中靠前而在普通房源中靠后的关键词的频度进行人为的加权操作。这样一来，那些在优质房源和普通房源中差异化出现的关键词就能够被真正提取出来供房东参考。

从下图的比较中可以看到，初步统计得到的关键词和频度校正后的关键词确实存在一定的区别。例如通过比较校正前后的结果可知：位于天坛和王府井周围的房源其实比位于天安门和故宫的房源更具吸引力。因此，使用校正后的关键词来指导房东的经营管理策略更具有实际意义。

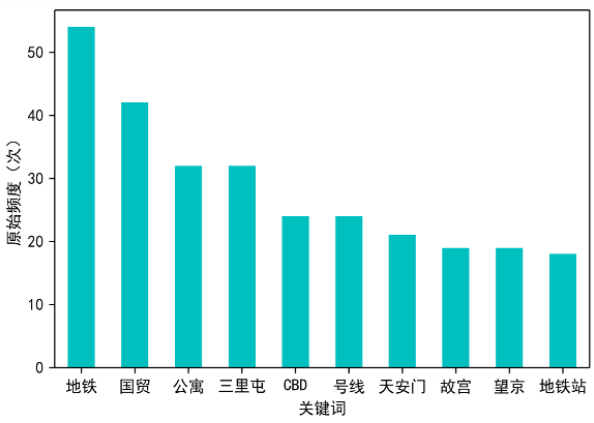

图 5.16(a) 北京市 Airbnb 优质房源关键词统计（频度校正前）

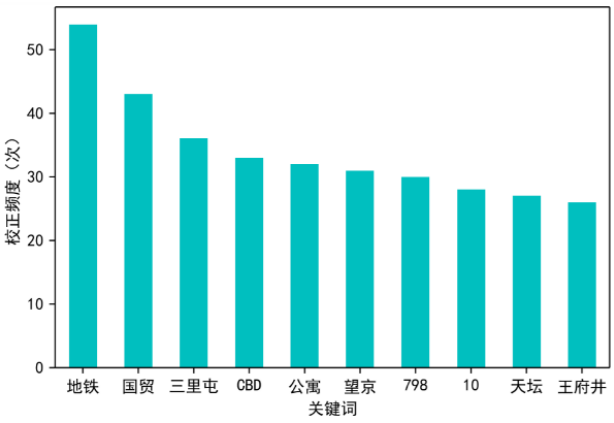

图 5.16(b) 北京市 Airbnb 优质房源关键词统计（频度校正后）

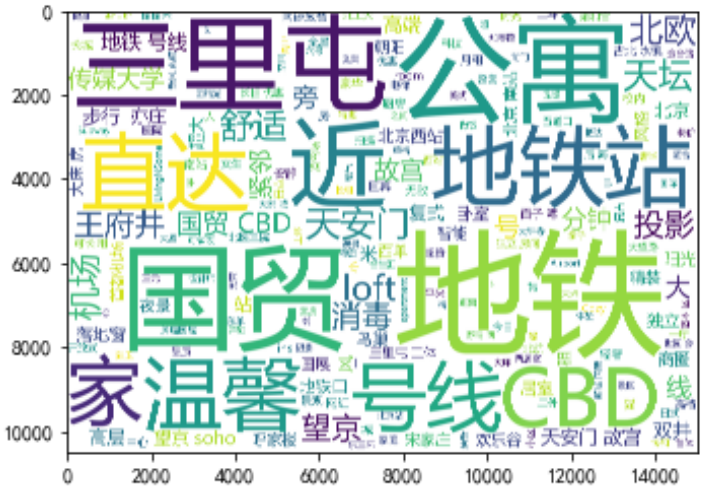

图 5.17 北京市 Airbnb 优质房源关键词云

6. 以能否即时预订为代表的一般属性对房源质量影响不大。

在大多数人的固有印象中，都会认为能够即时预定的房源更能够获得顾客的青睐。但统计结果表明：不允许即时预定的房源也能够以很高的概率成为优质房源。这可能是因为使用 Airbnb 的旅客更倾向于“计划先行”的出游，所以房源是否允许即时预定的影响也就不是那么显著了。同样的，根据频繁项集和关联规则对 amenities 属性单独进行分析可得：是否提供 Wifi、是否拥有取暖器、是否提供衣架等均为此类一般属性。

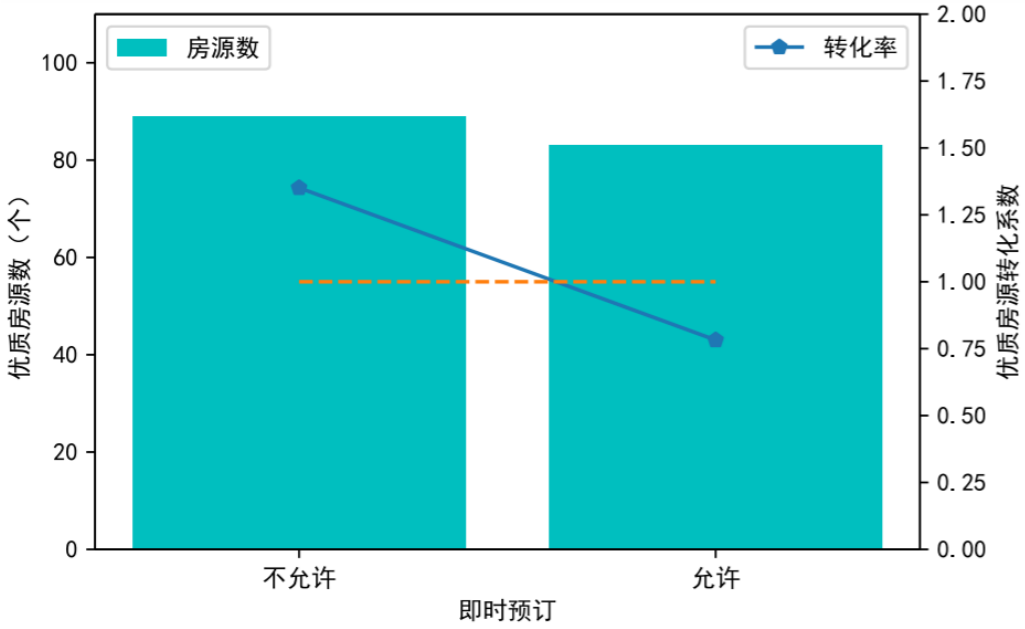

图 5.18 北京市 Airbnb 优质房源预定条件转化情况

5.3 房东分析——什么样的房东创收快？

5.3.1 结论

1. 相较于企业房东，个人房东能够为顾客提供更好的服务质量。
2. 评论回复率在大于 85%后，对进阶成为超级房东效果不显著。
3. 进行身份认证的房东更容易成为超级房东。

5.3.2 分析证明

1. 超级房东的诞生受多个条件共同影响，没有决定性因素。

从房东基本属性的相关性矩阵可以看到：各个域之间的相关性均不明显。这说明超级房东的行为模式的确比优质房源的特点更难发现和挖掘。但是依然可以从某些属性出发进行探索，总结进阶成为超级房东的基本行为准则。

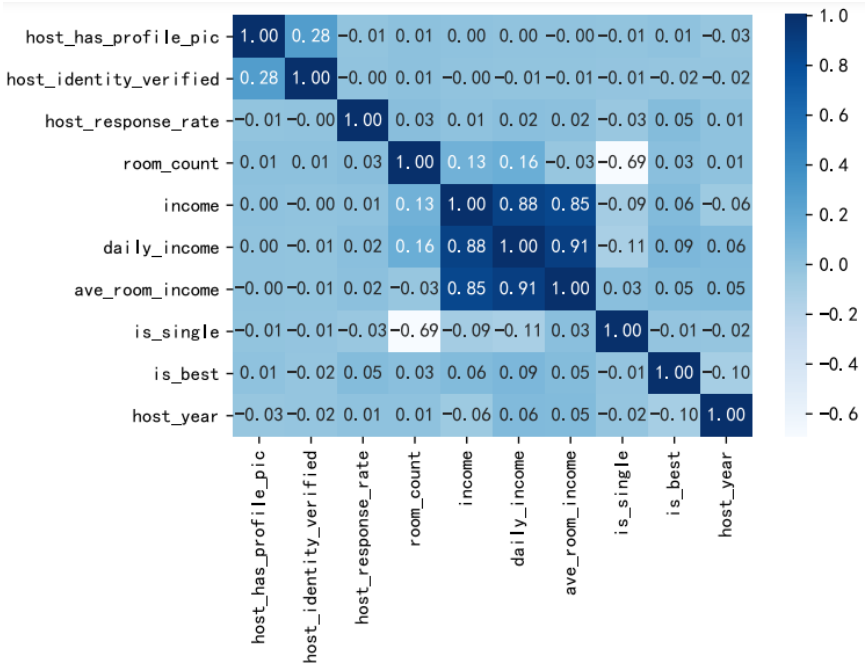

图 5.19 北京市 Airbnb 房东基本属性相关性矩阵

2. 相比于企业房东，个人房东能够提供更高质量的服务。

对于个人房东而言，主打服务质量，这一点可以从图 5.20 中看出——个人房东更能够将自己手中的房源打造成为优质房源。

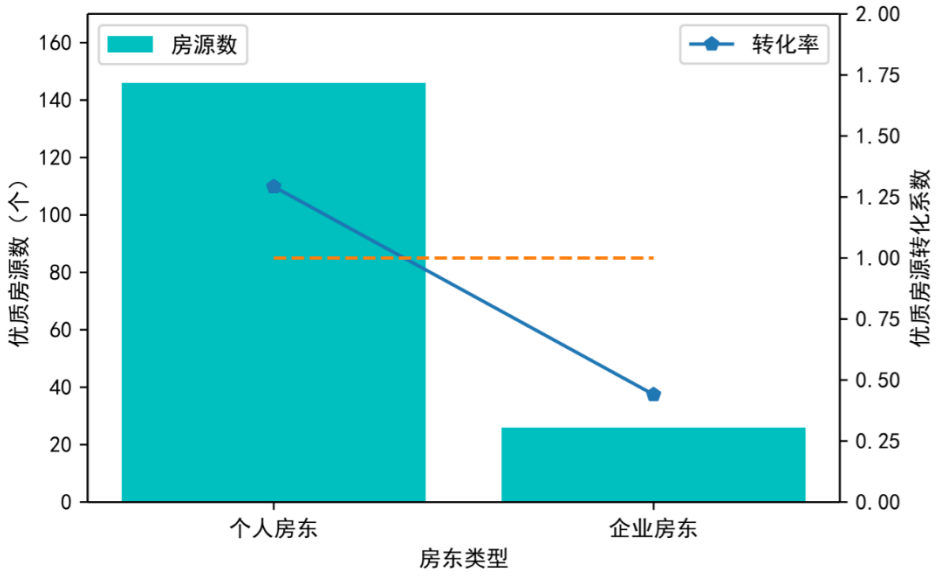

图 5.20 北京市 Airbnb 超级房东类型转化情况

3. 不用过度追求评论回复率，85%就足以保证房东的服务质量。

大多数人通常会认为房东的评论回复率和优质房源的产生之间是具有强相关性的——房东越积极地回复评论，说明这位房东越热心，又有谁不喜欢热情好客的房东呢？

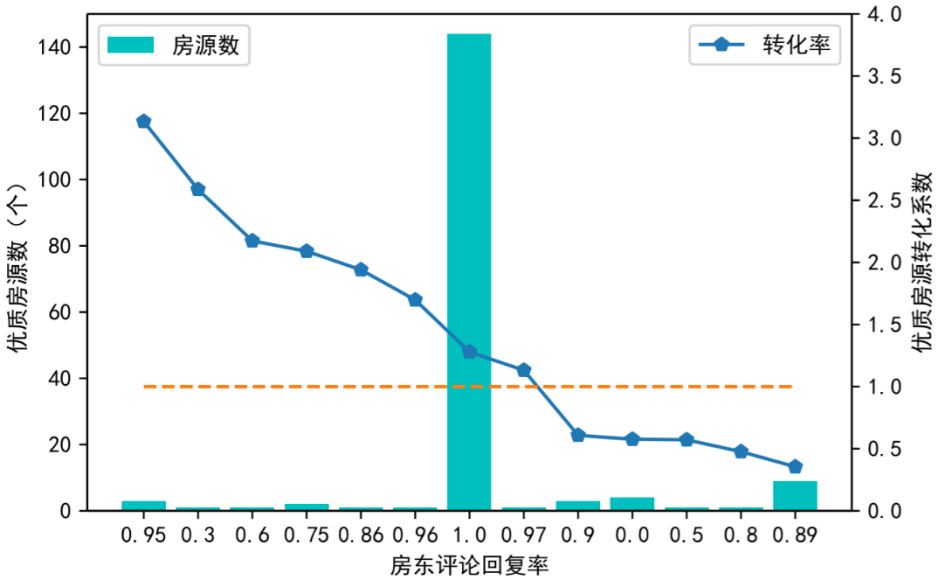

图 5.21 北京市 Airbnb 超级房东评论回复率转化情况

但从统计事实来看，这样的认知并不完全正确。具体来说，提高评论回复率对房东带来的收益存在边际递减效应。从上图的优质房源转换率中可以看到：大多数情况下，房东只需要对 85%的评论进行回复就已经能够实现比较理想的优质房源转化效果了。进一步提高回复率并不能够在房东的进阶之

路上留下浓墨重彩的一笔。因此，房东可以把回复评论的精力匀出一部分在其他方面下功夫，对成为超级房东或许会更有帮助。

4. **身份认证是成为超级房东的必经之路。**

如果按照之前看待转化系数思路直接对下图展开分析的话，很有可能会得到“不认证的房东更容易成为超级房东”这样无比滑稽的结论。事实上，数据分析的过程并不是一成不变的。前文之所以从转化系数出发对各个属性的重要性进行分析，是因为绝对数量上没有出现下图中这样极端情况：未认证房东个数甚至没有认证房东的零头多。换句话说，这里未认证房东并不能够被称为有效数据，是优质房源统计数据中的“噪声”，这些数据是正确的，但是因为过于少见而并不具有统计意义。

为了验证上述理论，将上海市超级房东进行统计发现并不存在未认证的情况。因此，如果对噪声的转化系数进行分析，那么得到的结论自然不可能是正确的。此时，可以认为超级房东均进行了身份验证，故进行身份验证也是房东进阶过程中的必经之路。

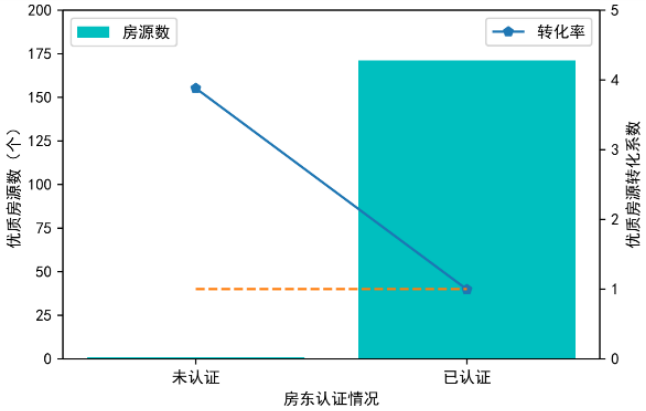

图 5.22(a) 北京超级房东认证转化情况

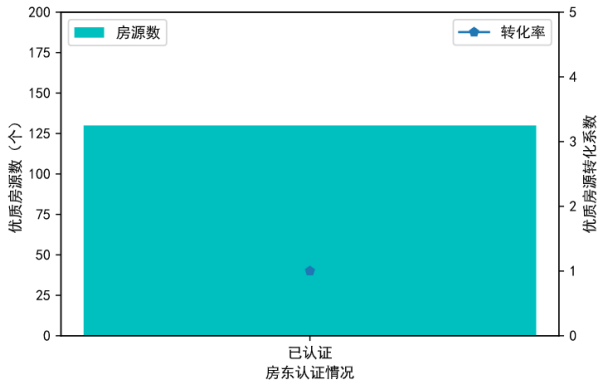

图 5.22(b) 上海超级房东认证转化情况

5.4 城市对比——如何因地制宜管理房源？

5.4.1 结论

1. 对于以盈利为首要目标的投资者来说，选择北京大概率能够获得更高的收益。
2. 对于企业房东来说，上海可能具有更高的回报率。

3. 北京市的房东在经营时应重点关注房源的地理位置，而上海市的房东更该在服务质量上精益求精。

5.4.2 分析证明

1. 北京市房源日均收入高于上海市，在 2020 年约为 1.75 倍。

从下图中可以看出，北京和上海在 Airbnb 进驻的十年基本具有相同的发展趋势。就房源日平均收入来说，上海在 2020 年突破了 80 元，而北京是 140 元。对于以盈利为首要目标投资者来说，北京可能是一个更有吸引力的选择。

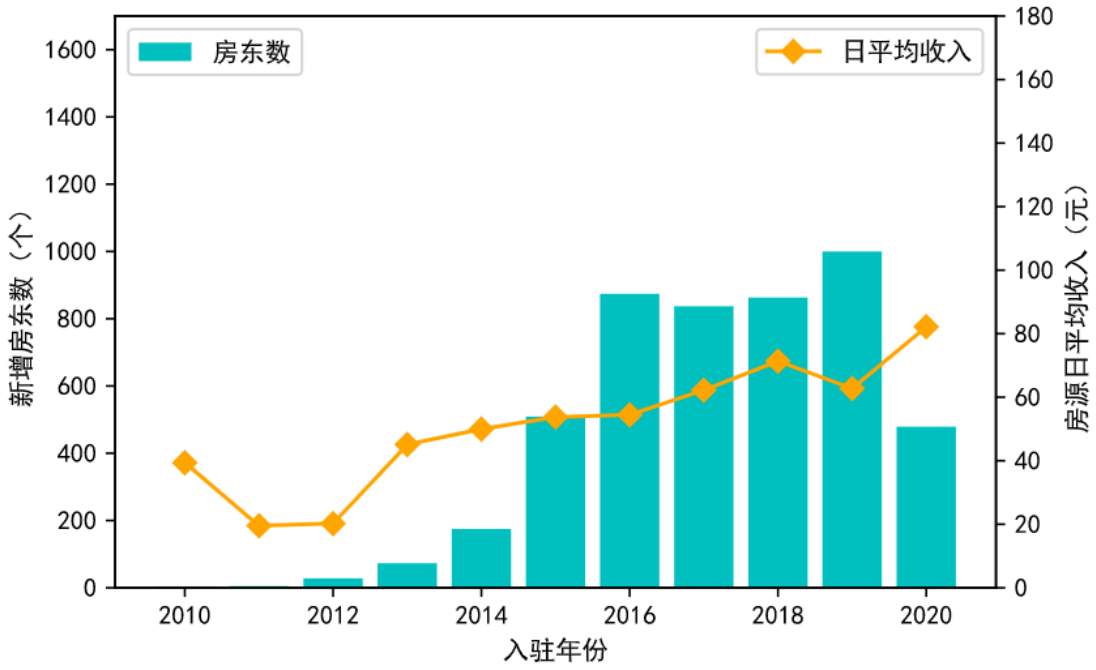

图 5.23 上海市 Airbnb 房东入驻及收入情况统计

2. 上海市的企业房东具有更强的盈利能力，日均收入约为北京市的 2.2 倍。

与北京市个人房东在房均收入上远超企业房东不同，上海市的企业和个人房东的房均收入差异并不大。上海市的企业房东在日均收入上远超上海的个人房东和北京市的两类房东。综上，上海是一个值得企业房东考虑的选择。

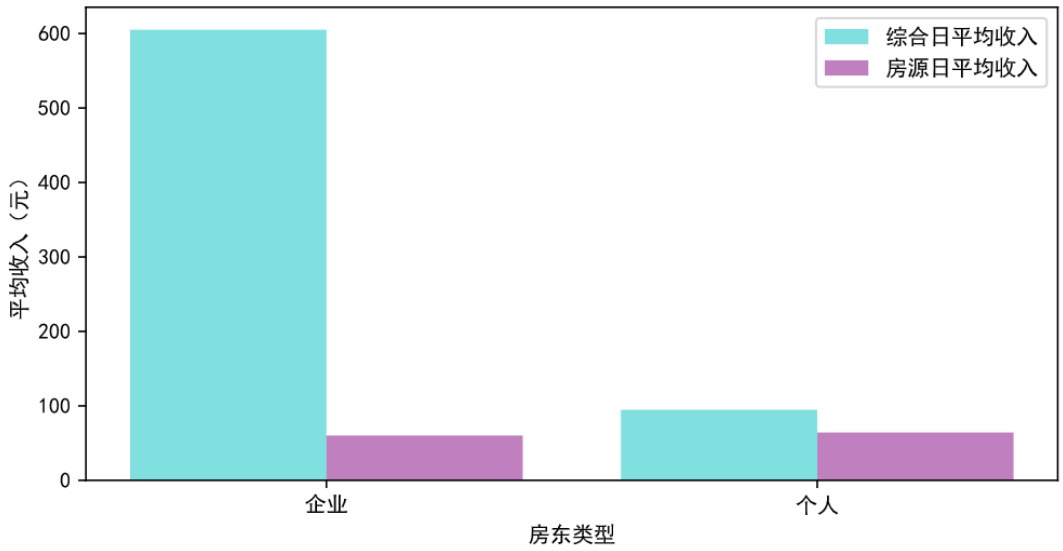

图 5.24 上海市 Airbnb 房东类型及收入情况统计

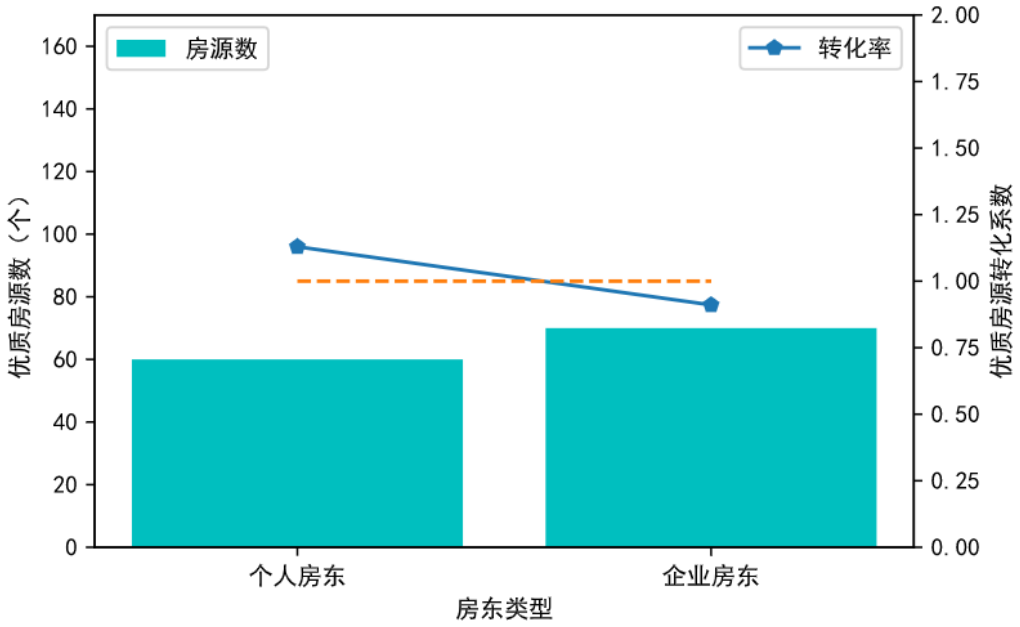

图 5.25 上海市 Airbnb 超级房东类型转化情况

3. 北京市房源经营重在地理位置的选择，上海市则重在服务质量。

从上海市优质房源的关键词可以看出，前十中只有“迪士尼”与地理位置相关，而北京有国贸、三里屯、望京、798、天坛、王府井总共 6 个。因此，选择在上海经营的房东需要更加注意服务质量，而选择北京的投资者则应重点关注房源的地理位置。

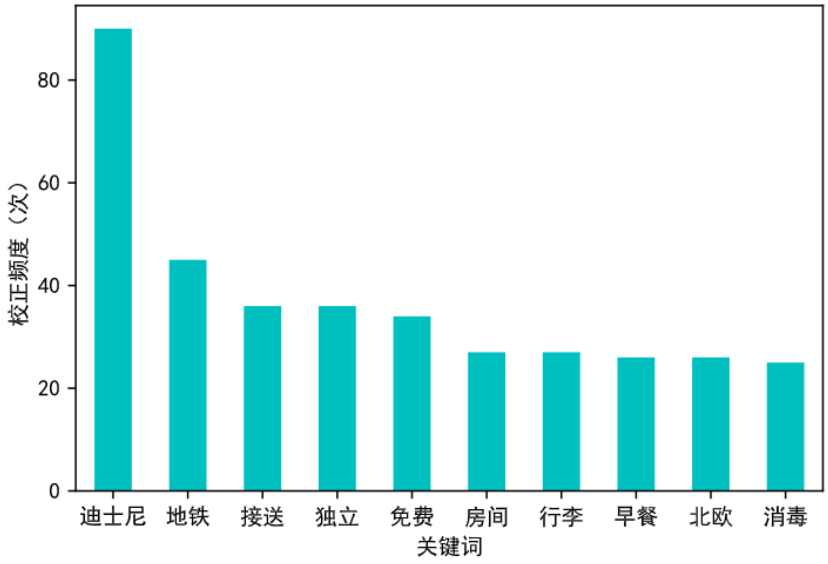

图 5.26 上海市 Airbnb 优质房源关键词统计（频度校正后）

6. 结论

本项目通过对 Airbnb 官方提供的房源和房东信息进行统计分析，说明了该平台强大的盈利能力，且值得广大房东和投资者的关注和选择；同时找到了优质房源的三条基本规律供投资者参考，旨在最大程度减小运营经验的获得成本；进一步对房东行为模式进行了分析，为进阶超级房东提供了可靠的行为准则；最后通过对比北京和上海两座大型城市的民宿运营情况，揭示了因地制宜管理房源的潜在规律。综上所述，希望相关探索成果能够为房东和投资者提供具有参考意义和实用价值的方法和结论，营造协作共赢的民宿租赁环境。

附录

A. 数据清洗代码

```
import os
import requests
import pandas_profiling
import pandas as pd


def load_data(filename: str) -> pd.DataFrame:
    data_df = pd.read_csv(filename)
    return data_df

def get_neighbourhood(room_info: pd.core.series.Series) -> str:
    param = {
        'key':
        'xxxx',
        'location':
        '{:.3f},{:.3f}'.format(room_info['longitude'], room_info['latitude']
),
    }
    response = requests.get('https://restapi.amap.com/v3/geocode/regeo',
                            params=param)
    return response.json()['regeocode']['addressComponent']['district']

def exchange_price(room_info: pd.core.series.Series) -> float:
    return float(room_info['price'].replace('$', '').replace(',', '')) * 6.5
3

def clean_room_data(room_df: pd.DataFrame) -> pd.DataFrame:
    drop_list = ['bathrooms']

    if os.path.exists('./data/clean/room.csv'):
        return load_data('./data/clean/room.csv')
    else:
        # drop meaningless attr
        room_df = room_df.drop(drop_list, axis=1)
        # specify neighbourhood as district
        room_df = room_df[room_df['neighbourhood'].notnull()]
        room_df = room_df[(room_df['neighbourhood'].str.find('Beijing') != -
1)
```

```
                        | (room_df['neighbourhood'].str.find('北京') != -1)]
        room_df['neighbourhood'] = room_df.apply(
            lambda x: get_neighbourhood(x), axis=1)
        room_df['price'] = room_df.apply(lambda x: exchange_price(x), axis=1)
        room_df = room_df[room_df['neighbourhood'].str.find('区') != -1]
        # filter illegal data
        room_df = room_df[(room_df['number_of_reviews'] != 0) & \
                          (room_df['review_scores_rating'].notnull()) & \
                          (room_df['name'].str.find('测试') == -1) & \
                          (room_df['name'].str.find('下架')==-1) & \
                          (room_df['name'].str.find('不能租')==-1) & \
                          (room_df['name'].str.find('下线')==-1)]
        room_df.to_csv('./data/clean/room.csv', index=False)
        pandas_profiling.ProfileReport(room_df).to_file(
            './data/profile/room_clean.html')
    return room_df

def clean_host_data(room_df: pd.DataFrame,
                    host_df: pd.DataFrame) -> pd.DataFrame:
    if os.path.exists('./data/clean/host.csv'):
        return load_data('./data/clean/host.csv')
    else:
        valid_host_id = room_df['host_id']
        host_df = host_df[host_df['host_id'].isin(valid_host_id)]
        host_df = host_df.drop_duplicates()
        host_df.to_csv('./data/clean/host.csv', index=False)
        pandas_profiling.ProfileReport(host_df).to_file(
            './data/profile/host_clean.html')
    return host_df

def clean_review_data(room_df: pd.DataFrame,
                      review_df: pd.DataFrame) -> pd.DataFrame:
    return review_df

def split_raw_data() -> pd.DataFrame:
    room_attr_list = [
        'room_id', 'host_id', 'neighbourhood', 'name', 'latitude', 'longitude',
        'room_type', 'accommodates', 'bedrooms', 'bathrooms', 'beds', 'price',
```

```
        'amenities', 'instant_bookable', 'first_review', 'last_review',
        'number_of_reviews', 'review_scores_rating'
    ]
    host_attr_list = [
        'host_id', 'host_name', 'host_since', 'host_has_profile_pic',
        'host_identity_verified', 'host_response_rate'
    ]

    if os.path.exists('./data/raw/room.csv') and \
       os.path.exists('./data/raw/host.csv') and \
       os.path.exists('./data/raw/reviews.csv'):
        room_df = load_data('./data/raw/room.csv')
        host_df = load_data('./data/raw/host.csv')
        review_df = load_data('./data/raw/reviews.csv')
    else:
        arb_df = load_data('./data/raw/listings-detail.csv').rename(
            columns={'id': 'room_id'})
        review_df = load_data('./data/raw/reviews.csv')

        # changes on room_df won't influence arb_df
        room_df = arb_df[room_attr_list]
        host_df = arb_df[host_attr_list]
        room_df.to_csv('./data/raw/room.csv', index=False)
        host_df.to_csv('./data/raw/host.csv', index=False)

        # generate raw data information
        pandas_profiling.ProfileReport(room_df).to_file(
            './data/profile/room_raw.html')
        pandas_profiling.ProfileReport(host_df).to_file(
            './data/profile/host_raw.html')
        pandas_profiling.ProfileReport(review_df).to_file(
            './data/profile/review_raw.html')
    return room_df, host_df, review_df

def eda():
    room_df, host_df, review_df = split_raw_data()
    room_df = clean_room_data(room_df)
    host_df = clean_host_data(room_df, host_df)

if __name__ == '__main__':
    split_raw_data()
```

B. 房源数据整理代码

```
import pandas as pd

# CITY = 'beijing'
CITY = 'shanghai'

room_df = pd.read_csv('./data/{}/room.csv.bak'.format(CITY))
host_df = pd.read_csv('./data/{}/host.csv.bak'.format(CITY))

def filter_noise(raw_room_df:pd.DataFrame):
    room_df = raw_room_df.copy(deep=True)
    room_df = room_df.drop(['amenities'], axis=1)
    drop_index_list = room_df[room_df['price']==0].index.tolist()
    room_df = room_df.drop(drop_index_list)
    # three sigma principle
    neighbourhood_group = room_df.groupby('neighbourhood')
    mean_price = neighbourhood_group['price'].mean()
    std_price = neighbourhood_group['price'].std()
    up_lim = mean_price +  3 * std_price
    low_lim = mean_price - 3* std_price
    valid_room_df_list = []
    for name, room in neighbourhood_group:
        valid_room_df_list.append(room_df[(room_df['neighbourhood']==name) & (room_df['price'] <= up_lim[name]) & (room_df['price'] >= low_lim[name])])

    room_df = pd.concat(valid_room_df_list)
    return room_df

def format_room(raw_room_df:pd.DataFrame):
    room_df = raw_room_df.copy(deep=True)
    room_df = room_df.fillna(value=0)
    # fill blanks before chang column type
    room_df['bedrooms']= room_df['bedrooms'].astype(int)
    room_df['beds']=room_df['beds'].astype(int)
    return room_df

def get_best_room_df(raw_room_df:pd.DataFrame):
    room_df = raw_room_df.copy(deep=False)
    review_num_lim = room_df['number_of_reviews'].quantile(0.9)
    review_score_lim = room_df['review_scores_rating'].quantile(0.9)
```

```
    best_room_df = room_df[(room_df['number_of_reviews']>=review_num_lim) & (room_df['review_scores_rating']>=review_score_lim)]
    return best_room_df

def preprocess_room(raw_room_df:pd.DataFrame):
    room_df = raw_room_df.copy(deep=True)
    room_df = filter_noise(room_df)
    room_df = format_room(room_df)
    best_room_df = get_best_room_df(room_df)
    return room_df, best_room_df

room_df, best_room_df = preprocess_room(room_df)

room_df.to_csv('./data/{}/room.csv'.format(CITY),index=False)

best_room_df.to_csv('./data/{}/best_room.csv'.format(CITY),index=False)

room_df.info()
```

C. 房东数据整理代码

```
import pandas as pd
from datetime import datetime


# CITY = 'beijing'
CITY = 'shanghai'

room_df = pd.read_csv('./data/{}/room.csv.bak'.format(CITY))
host_df = pd.read_csv('./data/{}/host.csv.bak'.format(CITY))

def stats_income(room_df: pd.DataFrame, raw_host_df: pd.DataFrame):
    host_df = raw_host_df.copy(deep=True)
    for host_id in host_df['host_id']:
        temp_room_df = room_df[room_df['host_id']==host_id]
        room_cnt = len(temp_room_df)
        total_income = (temp_room_df['price'] * temp_room_df['number_of_reviews']).sum()
        time_span = (datetime(2021,2,22) - pd.to_datetime(host_df[host_df['host_id']==host_id]['host_since'].values[0])).days
        host_df.loc[host_df['host_id']==host_id, 'room_count'] = room_cnt
        host_df.loc[host_df['host_id']==host_id,'income'] = total_income
```

```
        host_df.loc[host_df['host_id']==host_id,'daily_income'] = total_income / time_span
        host_df.loc[host_df['host_id']==host_id,'ave_room_income'] = total_income / (time_span*room_cnt)
    return host_df

def tag_host(raw_host_df:pd.DataFrame):
    host_df = raw_host_df.copy(deep=True)
    host_df['is_single']=host_df['room_count'].map(lambda x: 't' if x < 5 else 'f')

    best_room_df=pd.read_csv('./data/beijing/best_room.csv')
    best_host_id_list = best_room_df['host_id'].values.tolist()
    host_df['is_best'] = host_df['host_id'].map(lambda x: 't' if x in best_host_id_list else 'f')
    return host_df

def format_host(raw_host_df:pd.DataFrame):
    host_df = raw_host_df.copy(deep=True)
    host_df['host_response_rate'] = host_df['host_response_rate'].map(lambda x: x if pd.isna(x) else round(float(x[:-1])/100,2))
    host_df = host_df.fillna(round(host_df.mean(), 2))
    host_df = host_df.fillna({'host_identity_verified': 'f', 'host_name':'_', 'host_has_profile_pic': 'f'})
    host_df[['room_count']]=host_df[['room_count']].astype(int)
    for host_id in host_df[pd.isna(host_df['host_since'])]['host_id']:
        host_df.loc[host_df['host_id']==host_id, 'host_since']=room_df[room_df['host_id']==host_id].sort_values(by='first_review')['first_review'].values[0]
    return host_df

def preprocess_host(room_df: pd.DataFrame, raw_host_df: pd.DataFrame):
    host_df = raw_host_df.copy(deep=True)
    host_df = stats_income(room_df, host_df)
    host_df = tag_host(host_df)
    host_df = format_host(host_df)
    return host_df

host_df = preprocess_host(room_df, host_df)

host_df.to_csv('./data/{}/host.csv'.format(CITY), index=False)

```