参赛学校： 重庆科技大学

论文题目： 基于深度学习的阿联酋房租预测研究

参赛队员： 王静、李思成、吴雯

指导教师： 陈照辉

**2024年（第十届）全国大学生统计建模大赛**

目录

[摘要 2](#_Toc25515)

[第一章 绪论 3](#_Toc3227)

[（一）研究背景 3](#_Toc8881)

[（二）研究意义 3](#_Toc5805)

[（三）研究现状 4](#_Toc7467)

[（四）研究思路 4](#_Toc4813)

[第二章 指标的选取与数据处理 6](#_Toc17731)

[（一）数据的来源 6](#_Toc10174)

[（二）指标的选取 6](#_Toc14058)

[（三）数据预处理 7](#_Toc14588)

[第三章 基本数据集分析 8](#_Toc27060)

[（一）租金价格分布 8](#_Toc8625)

[（二）对租金价格等级进行分类 8](#_Toc3712)

[（三）每平方米的租金价格分布 9](#_Toc28719)

[（四）卧室数量对租金的影响 10](#_Toc24446)

[（五）浴室数量对租金的影响 10](#_Toc28265)

[（六）总面积对租金的影响 11](#_Toc7509)

[（七）房产类型对租金的影响 11](#_Toc26365)

[（八）位置对房租的影响 12](#_Toc2670)

[（九）城市对租金的影响 14](#_Toc21456)

[（十）房产的家具状态对租金的影响 15](#_Toc32183)

[（十一）挂牌天数与出租需求分布图 16](#_Toc19022)

[（十二）房产的用途租金分布 16](#_Toc12374)

[（十三）挂牌出租的日期对房价的影响 17](#_Toc23233)

[第四章 简单基本模型 18](#_Toc17961)

[（一）线性回归(Linear Regression) 18](#_Toc870)

[（二）决策树(Decision Tree) 18](#_Toc20227)

[（三）XGboost(（eXtreme Gradient Boosting） 19](#_Toc7462)

[（四）模型预测结果 19](#_Toc12030)

[（五）特征权重 20](#_Toc5919)

[第五章 深度学习 22](#_Toc3694)

[（一）神经网络（Neural Networks，NN） 22](#_Toc16221)

[（二）长短时记忆网络（Long Short-Term Memory，LSTM） 26](#_Toc13863)

[第六章 总结 31](#_Toc2677)

[致谢 32](#_Toc9014)

[参考文献 33](#_Toc6369)

[附录 34](#_Toc13642)

摘要

在全球经济背景下，房地产市场的稳定和发展对各国都至关重要。房租作为房地产市场的核心指标，反映了居民的生活成本和房地产市场的供需关系。阿联酋作为中东地区的经济强国，其房地产市场对全球具有示范意义。

本研究针对阿联酋房地产数据集进行了房租预测，旨在为政府和企业提供准确的决策依据。通过使用Python技术获取了阿联酋多个城市的出租物业数据，包括地址、租金、房型等特征。在数据进行了清洗和预处理后，进行了可视化分析，探究了租金价格分布、卧室数量、浴室数量、总面积等因素对租金的影响。对此，本文进一步建立了决策树、线性回归、XGBoost和神经网络等模型，使用交叉验证方法评估了模型的预测效果，结果显示，深度学习中的LSTM模型表现最优，其MSE、RMSE、SMAPE值均较小，R2拟合度也较高，能合理预测阿联酉的房租预测。特征重要性分析显示，卧室数量是最重要的特征。本研究不仅为政府制定房地产政策提供了有力的数据支持，也为投资者和开发商在房地产市场中的决策提供了科学的依据。同时，对于广大租房者来说，本研究的结果可以帮助他们更好地了解市场行情，从而做出更为明智的租房选择。

**关键词：**房租预测 XGBoost 神经网络 深度学习 人工智能

# 第一章 绪论

## （一）研究背景

房地产在全球每个国家都至关重要。它是民生的保障、国民经济的支柱、财富的象征、大类资产配置的核心、金融危机的策源地。随着全球化的深入发展，全球房地产市场已成为影响全球经济的重要力量。近年来，全球经济环境的复杂多变、人口迁移的加剧以及科技创新的推动，使得全球房地产市场呈现出前所未有的活跃态势。然而，这种活跃态势背后也隐藏着诸多不确定性因素，如经济波动、政策调整、社会变革等，都对房地产市场的稳定性和可持续发展提出了挑战。

在全球房地产市场的背景下，中国房地产市场也经历了快速的发展。作为世界人口大国的国家之一，中国的城市化进程加速，房地产市场需求旺盛。特别是在过去几十年中，中国经济的快速发展和城市化进程的加速，为房地产市场提供了巨大的增长空间。然而，与此同时，中国房地产市场也面临着一些问题和挑战，如房价高涨、库存积压、供需失衡等，这些问题不仅影响了房地产市场的健康发展，也对社会经济稳定带来了挑战。

为了解决这些问题，我们需要对房地产市场进行深入的研究和分析。而在这个过程中，数据的作用至关重要。随着大数据和人工智能技术的不断发展，我们有许多的手段和方法来挖掘和利用房地产数据，为房地产市场的预测和决策提供更加精准和有效的支持。

在大数据与人工智能时代下，我们充分利用开源数据集获取了一个样本量丰富的阿联酋房地产数据库，该数据集包括了房价、房租、房屋类型、地理位置等多个维度的数据。通过机器学习和深度学习模型对这些数据进行分析，我们可以更加深入地了解房地产市场的运行规律和趋势，为政府和企业提供更加准确的决策依据，以此来更精准地把握中国房地产市场的运行状况，帮助我国更好地应对房地产市场的变化和挑战，促进房地产市场的平稳健康发展。

## （二）研究意义

本文针对阿联酋房地产数据集进行房租预测，有助于全球投资者和决策者更好地理解国际房地产市场的动态，其对社会、国家和国际在房地产经济上有一定的借鉴作用。

### 1.2.1在房地产市场方面

通过对阿联酋房租的预测研究，可以洞察阿联酋的房地产市场趋势，帮助阿联酋房地产市场参与者更好地理解市场供需关系。这对于当地的房地产开发商、投资者和中介机构等市场主体来说，具有重要的决策参考价值。该研究可以为其他国家和城市的房租市场研究提供实证参考，特别是使用大数据和机器学习深度学习进行市场预测方面，这为我国在房租价格波动和市场监管方面提供了新的视角和方法，有利于优化资源配置和提高市场效率。

### 1.2.2在国家政策制定方面

阿联酋作为中东地区的经济强国，我们研究阿联酋的房租预测可以为政府调整房地产政策提供依据，制定更为有效的市场调控政策。其房地产市场的发展经验对于中国来说具有一定的参考价值，这有助于中国政府借鉴国际经验，制定更加符合国情的房地产政策。

### 1.2.3在国际方面

与中国相比，阿联酋的房地产市场可能更加成熟和稳定，其在房地产市场风险管理方面可能积累了一定的经验，为全球房地产市场的健康发展提供了经验和教训。中国可以学习其先进的风险管理经验和技术手段，有助于中国在国际房地产市场中更好地定位自己，促进国际之间的经济交流和合作，推动经济社会和谐进步。

## （三）研究现状

### 1.3.1 国内研究现状

近年来，随着城市化进程的加速和人们居住需求的多样化，租房市场愈发繁荣，房租价格预测也成为了学术界和实践界关注的热点。高晨辉[1]对收集到的32278条房源租赁数据按照8：2的比例划分训练集和测试集，构建多元线性回归模型，随机森林和XGBoost，并比较了杭州市不同维度数据特征对最终预测效果的影响。结果表明，大数据采集和机器学习模型能够对租房租赁市场租金价格提供合理预测。夏静伶[2]运用Python软件从链家网站上爬取了武汉市2022年2月的24000条真实房源数据，对数据进行可视化分析，分析出住房的面积，所在地理位置等各种特征因素对租金的影响，同时也运用到多元线性回归和三种机器学习模型，其中得出结论：XGBoost模型误差最小，用来预测武汉市住房租金效果最好，最后对住房租赁市场的发展提出相关建议。

### 1.3.2 国外研究现状

国外研究学者Tom Engsted和Thomas Q. Pedersen[3]利用18个经合组织（OECD）国家1970年至2011年的年度和季度数据，房价租金比（Rent-to-Price Ratio）来预测房地产回报和租金增长，探讨了租金-房价比率作为预测变量的有效性。研究发现租金增长的预测模式在不同国家和时期之间存在很大的不稳定性，并且这种预测模式在名义或实际条件下测量时会有很大差异。Sujata Khandaskar、Chirag Panjwani等研究学者[4]利用回归分析方法来预测房价和租金，研究旨在提供一个更准确和可靠的预测系统，以帮助房主、租户和房地产专业人士做出明智的决策。Quang Truong、Minh Nguyen、Hy Dang和Bo Mei[5]利用改进的机器学习技术来预测房价,该论文将应用传统和先进的机器学习方法来调查几种先进模型之间的差异，并在回归模型实施中全面验证多种技术，为房价预测提供乐观的结果。Xiaojie Xu和Yun Zhang[6]使用神经网络来预测中国城市的房价，开发了一种简单、准确的模型，能够在不同阶段保持稳定的预测性能。

## （四）研究思路

本文对房产租金预测研究思路为以下六个部分：

第一步数据获取与预处理：利用Python爬取阿联酋房地产数据集，并进行数据清洗，删除异常值，以确保数据质量。同时，进行数据标准化处理，为建模提供可靠的数据基础。

第二步对特征进行选择与降维：采用相关性分析、主成分分析等方法，从原始数据中筛选出对房租影响较大的特征变量，以简化模型，降低计算复杂度。

第三步构建模型与参数优化：采用多种机器学习算法构建房租预测模型，包括线性回归、决策树、XGBoost、神经网络等，并设置合理的参数。同时，利用网格搜索等方法进行参数优化，以获得最优模型。

第四步评估与选择模型：采用交叉验证法对模型进行评估，选择误差小的模型，并分析模型的性能。同时，比较不同模型的预测效果，选择最适合阿联酋房租预测的模型。

第五步对结果进行分析与可视化：对模型结果进行可视化分析，解释模型预测效果，通过图表分析不同特征对房租的影响程度。

第六步对研究进行总结：总结本研究存在的不足，如样本量有限、特征选择不全面等，并提出未来研究方向。在后续研究中，可考虑扩大样本规模、引入更多特征、探索其他机器学习算法等，以提高预测准确性

# 第二章 指标的选取与数据处理

## （一）数据的来源

本文利用python爬取了阿联酋房地产的数据集，它全面概述了阿拉伯联合酋长国多个城市的出租物业列表，包括阿布扎比、迪拜、沙迦、阿治曼、哈伊马角、乌姆奎因和艾因。该数据集爬取自bayut.com，是数据分析师、数据科学家和研究人员探索阿联酋房地产趋势、租金定价模式或城市发展研究的宝贵资源。数据集中的每个条目都代表了一个出租物业列表，其中包含有关该物业的功能、租赁条款和具体位置的详细信息，可用于深入了解阿联酋的租赁市场动态。

数据集:

/dubai\_properties.csv

代码:

/analysis\_main.ipynb

/analysis\_DL.ipynb

/analysis\_ML.ipynb

模型(torch):

/pre\_models/MLP/\*.pth 、 /pre\_models/LSTM/\*.pth

## （二）指标的选取

合适的指标能够全面，准确地反映城市租金的影响因素，能为后续的建模分析提供有利的支持。本文保持着相关性原则，选取的指标与租金均有直接的关系，以便反应租金变化的内在规律，并且保持全面性原则，涵盖多个方面的影响因素进行理论分析。本文的具体指标见表2-1。

表 2-1 数据集的指标

|  |  |
| --- | --- |
| 指标 | 属性 |
| Address | 房产的详细地址 |
| Rent | 年租金价格 |
| Beds | 卧室数量 |
| Baths | 浴室数量 |
| Type | 房产类型（例如，公寓，别墅，顶层公寓） |
| Area\_in\_sqft | 房产的总面积（平方英尺） |
| Rent\_per\_sqft | 每平方米的租金价格，计算方法是租金除以面积 |
| Rent\_category | 根据阈值对租金价格进行的分类（低、中、高） |
| Frequency | 租金支付频率，一般为 "每年" |
| Furnishing | 房产的家具状态（带家具、不带家具） |
| Purpose | 挂牌的目的，通常为 "出租" |
| Posted\_date | 房源挂牌出租的日期 |
| Age\_of\_listing\_in\_days | 挂牌天数 |
| Location | 房源所在城市的具体位置 |
| City | 房产所在的城市 |
| Latitude | 房产的地理坐标 |

## （三）数据预处理

数据集包含了73742条数据，首先对数据进行清洗，对异常值进行删除，NaN视为缺失值，由于样本足够大，删除掉所有含有至少一个NaN值的行，保留了73023 条数据。本文对数据处理主要使用机器学习算法，还需对类别特征进行编码，将非数值类型的特征进行编码，保证标签编码分配一个唯一的整数。在后续建立模型中，为了避免梯度消失或梯度爆炸而影响神经网络的训练效率和效果，使用sklearn包中的归一化器进行归一化处理，它通过对每个特征减去其均值并除以其标准差来实现，保持特征之间的相关性。本文的房产租金的数据见图2-1, 首次查看数据后不再展示.ipynb文件在vsc中的状态。

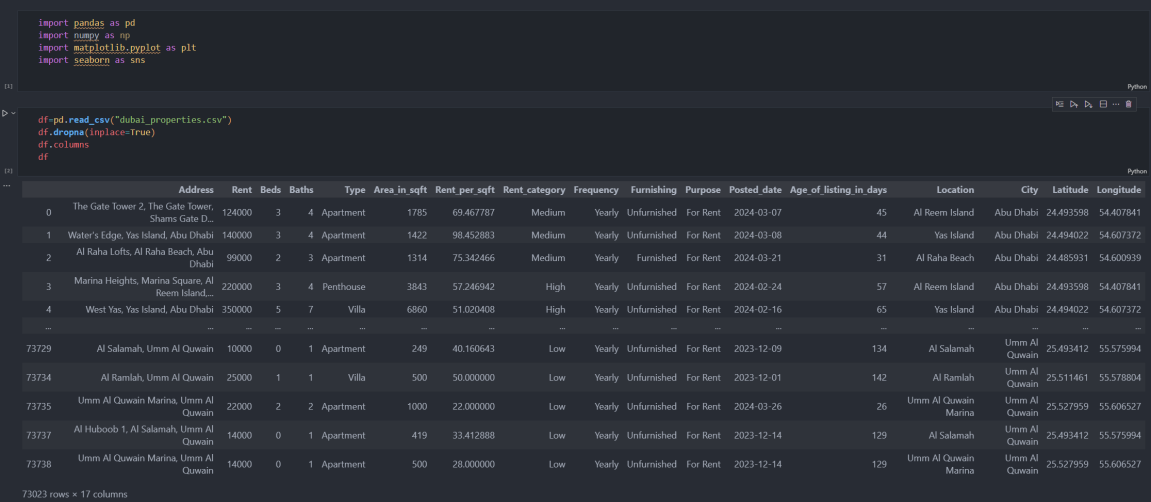


图 2-1 房产租金的数据

# 

# 第三章 基本数据集分析

## （一）租金价格分布

在大数据背景下，对租金价格分布进行深入分析和研究具有重要的现实价值和广泛的应用前景。阿联酉城市租金价格分布反映了城市租金市场的供需关系。不同地区的租金价格差异通常反映了该地区的经济发展水平、人口流动、基础设施完善程度以及房地产市场的活跃程度。因此，通过分析阿联酋房地产的数据集，我们可以更准确地把握市场的供需状况，为未来的市场预测和规划提供重要依据。政府可以分析租房价格分布来制定租金调控政策；投资者可以分析租金价格分布做出明治的投资决策；租房者也可以了解租金价格分布选择合适的租房地点和物业类型，来满足自身的生活需求和经济承受能力。

通过查询阿联酉这个城市73023条租房价格房源数据，对数据进行分析与处理后，绘制出价格分布直方图与箱线图。从图3-1中可看出，阿联酉国家的租金价格平均在105AED和106AED时候频数最多，可得出这两个价格对阿联酉人民最易接受的价格。其余租金所对应的租房频数有所差异可能与该房屋的地理环境，房屋类型，是否重新装修等因素有关。

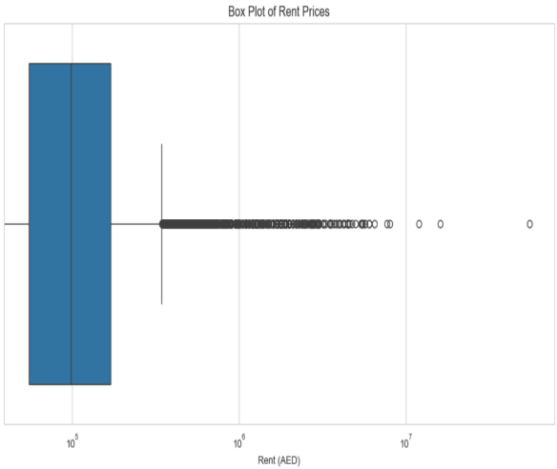
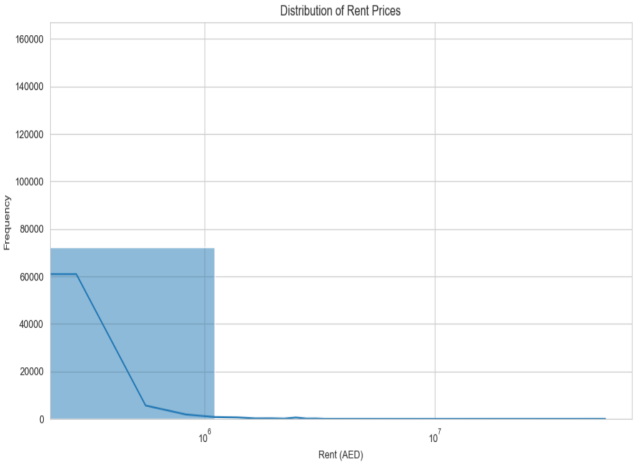


图 3-1租金价格分布

## （二）对租金价格等级进行分类

人们在租房过程中，往往会选择价格实惠，又有质量的房屋类型，因此价格的分类在本文中也是至关重要的部分。从获取的这73023条数据中，根据阈值把租金价格分为高，中，低这三个等级，具体的租金价格等级见图3-2。图表中的租房频数没有太大的差异，可得出结论：租房价格的等级对租房频数的影响几乎没有影响，三者之间没有太大差异，可能与房屋类型，房屋地理位置，房屋是否装修等因素有关。

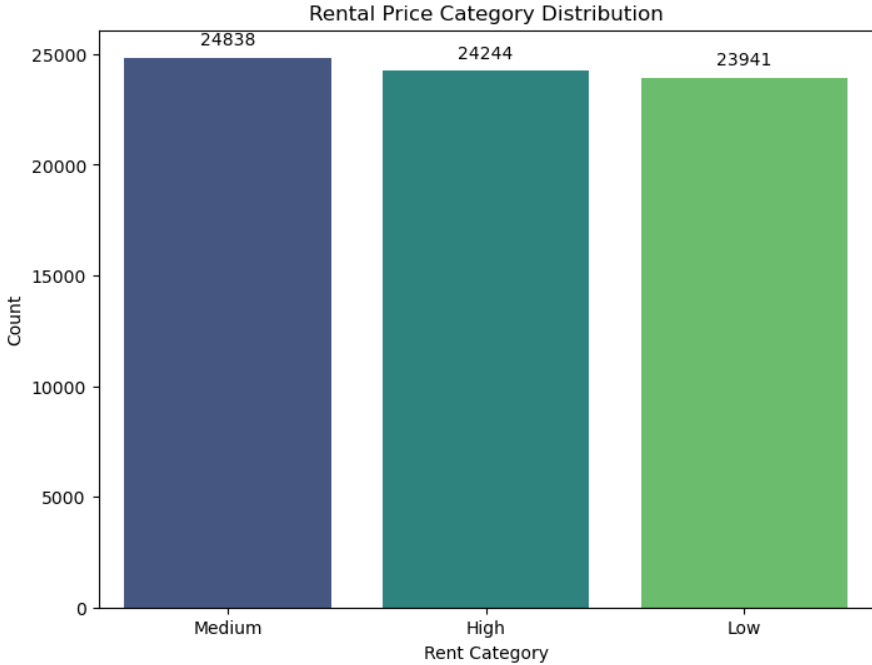
****

图 3-2租金价格等级分布

## （三）每平方米的租金价格分布

在租房群体租房过程中，面积是不可磨灭的重要因素，侧面来讲，每平方米的价格也是人们竞相关注的重点。在阿联酉城市租房数据中，每平方英尺的价格决定了租房人次的多与少。根据图3-3的直方图可看出，租房群体对于租房的价格均在500 AED/m2以下。每平方米的价格超过500 AED/m2后，就不会再有人民去租房，这其中可能与居民的经济水平，收入水平等因素有关。

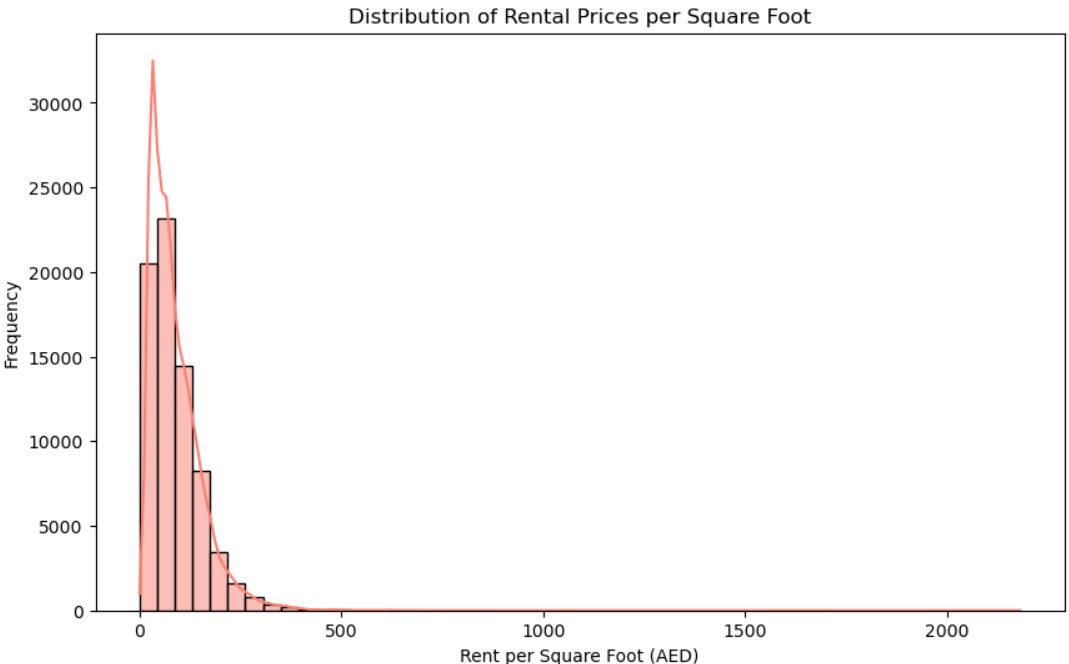


图 3-3每平方米的租金价格分布

## （四）卧室数量对租金的影响

数据中卧室数量主要集中在1到12个内，为了更佳分析卧室数量对租金的影响，对图表取log对数的图表见图3-4。在取对数后卧室数量与平均租金图表中，可看出，随着卧室数量的增加，平均租金逐步增长，但卧室数量在8，9，12时，平均租金有所下降，与整体趋势存在偏差，这可能与房屋类型，地理位置等因素导致的。

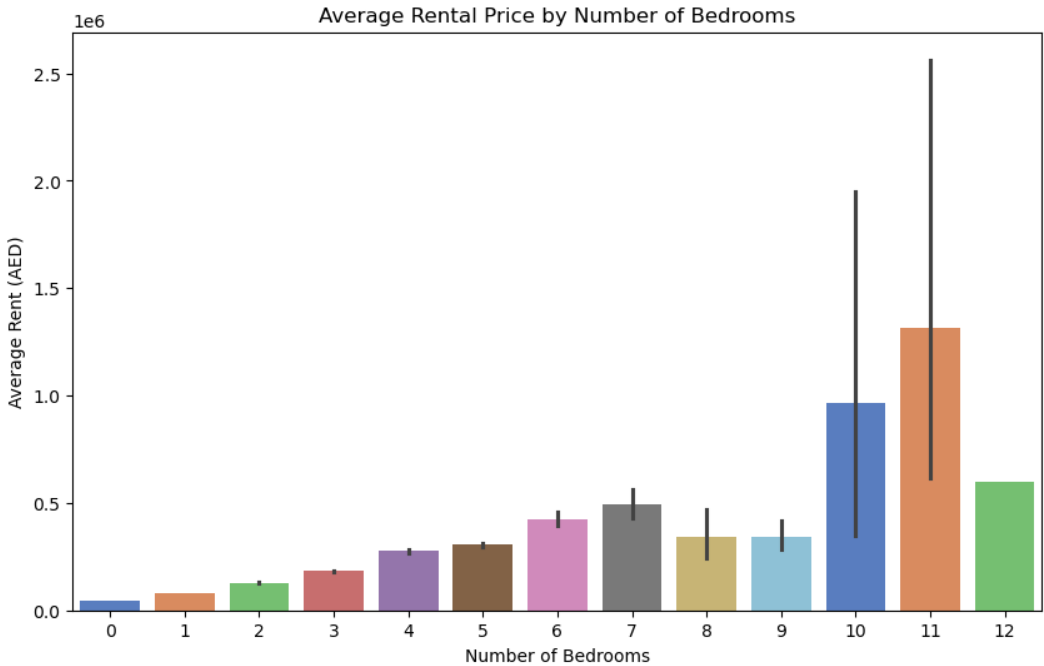


图 3-4卧室数量与租金的分布

## （五）浴室数量对租金的影响

随着租房群体更加追求个性化服务，对于房间里的配套设施要求也更加精炼。从浴室数量与租金箱线图（图3-5）可看出浴室数量在4-7个之间时，租金价格都相差不大，说明浴室数量在这个区间时价格都相对稳定。但在1-3个浴室数量和8-12个浴室数量之间时，租金的价格就会出现不断增长的趋势。根据图3-5大致差异可看出，对于不同浴室数量的租金价格尊在一定差异。可能与房屋的类型，或者与房屋的装修与维护情况有关。

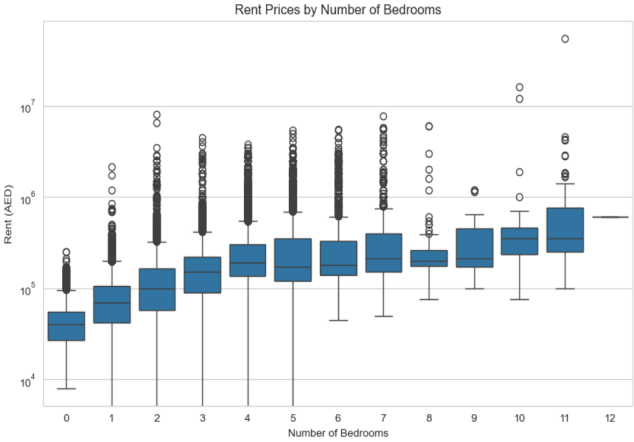


图 3-5 浴室数量与租金的分布

## （六）总面积对租金的影响

数据中房屋面积主要集中于0-50000平方米内，为了更细化租金与房屋的总面积关系，对图表取log对数，具体的图表见3-6。在取对数后的房屋租金与房屋总面积图表中，房屋总面积与房租呈正相关关系，即随着面积的增加，租金大致呈现上升趋势。但有些点的位置可能与整体趋势存在偏差，这可能是由于地理位置、房屋类型、是否带家具等因素影响导致的。

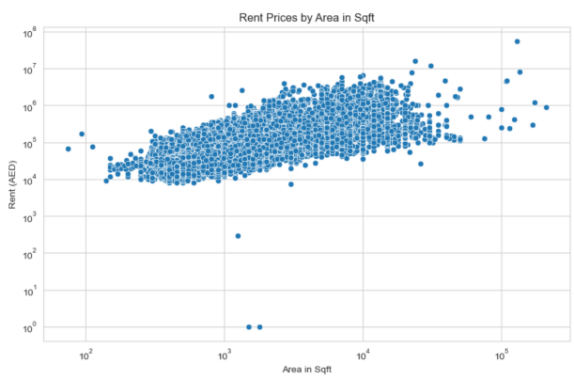
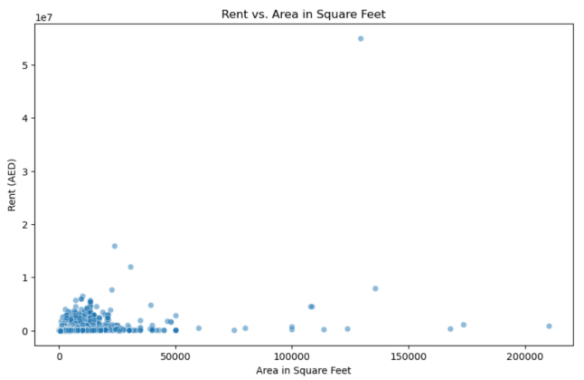


图 3-6总面积与租金的分布

## （七）房产类型对租金的影响

在数据集的物业类型中绘制柱状分布图，如图3-7-1所示，表中公寓（Apartment）的数量为56153套，是最受欢迎的物业类型，反映出公寓的多样性和价格范围，适合不同收入水平和居住需求的租户；别墅（Villa）的数量为12559套，需求比较稳定，具有一定的市场需求，适合寻求独立空间和隐私的家庭或高收入群体；联排别墅（Townhouse）的数量为3417套，需求较一般；酒店公寓（Hotel Apartment）的数量为461套，需求较低，消费对象可能更适合短期住宿或特定的居住需求群体，而顶层公寓（Penthouse）、别墅复合体（Villa Compound）、住宅楼层（Residential Floor）、住宅用地（Residential Plot）的数量在300套以下，在房租市场比较罕见，需求极低。

为了进一步观察物业类型与租金的关系，绘制不同房产类型与其对应的平均租金价格之间的柱状图，如图3-7-2所示。图中可以看到，住宅楼（Residential Building）的平均租金价格最高，达到280,0000 AED左右，远高于其他类型的房产，住宅楼层（Residential Floor）平均租金价格次之，达到140,0000 AED，而其他类型的房产住宅楼层（Residential Floor）、住宅用地（Residential Plot）等平均租金都相对较低，都在500,000 AED以下，反映出各种类型房产之间的平均租金存在显著差异。

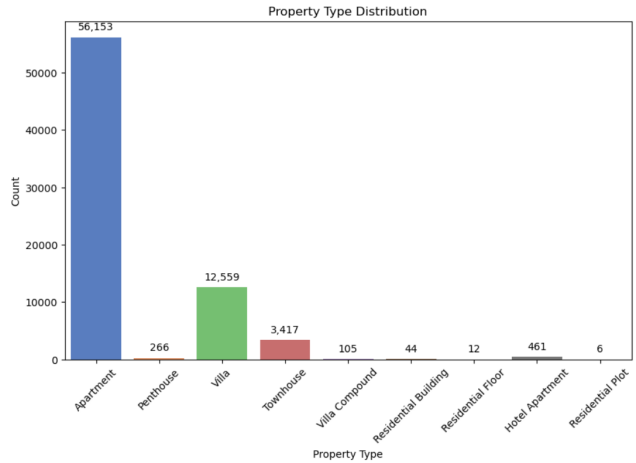


图 3-7-1物业类型分布

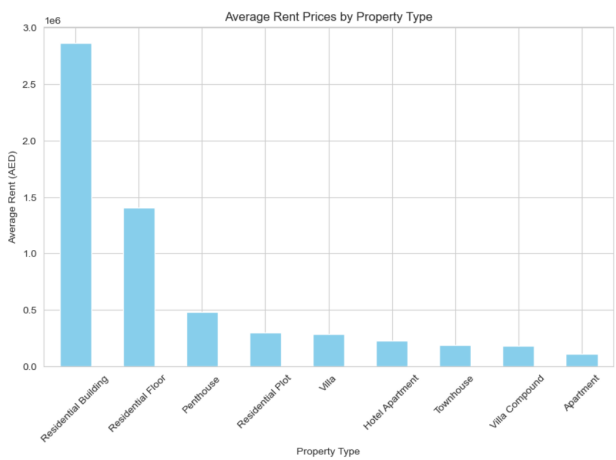


图 3-7-2 房产类型与租金的分布

## （八）位置对房租的影响

利用matplotlib绘制阿联酉的房产地理分布与各地理位置的租金分布散点图，见图3-8-1。

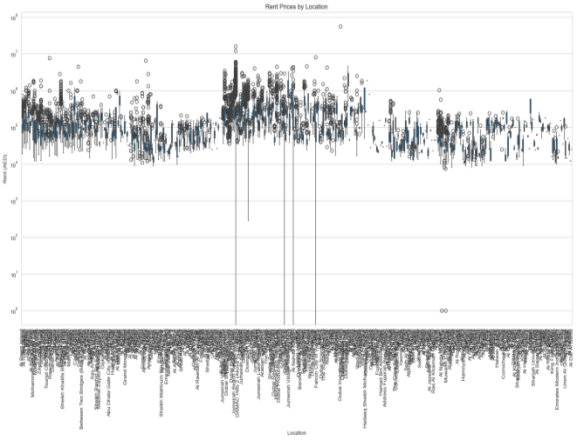
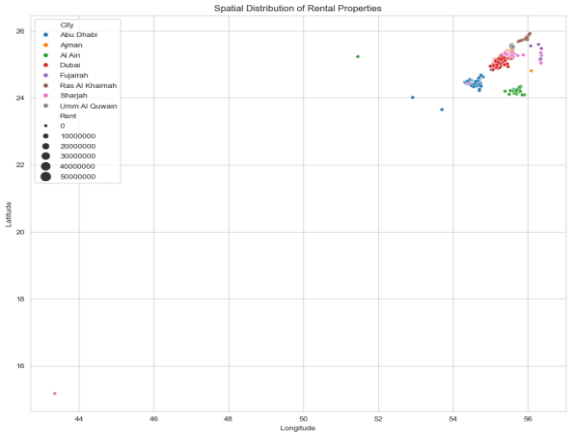


图 3-8-1 房产地理分布情况

由于地理分布较多，我们只显示租金中位数排名前20位的地点，见图3-8-2所示。由这个箱线图可知大多数地点的租金价格范围介于105AED到106AED，一些地点如Jumeirah Islands、EmiratesHills的租金中位数较高，而其他地点如Arabian Ranches、The Villa的租金中位数较低,从箱线图的长度可以看出，不同地点的租金价格存在一定的差异。

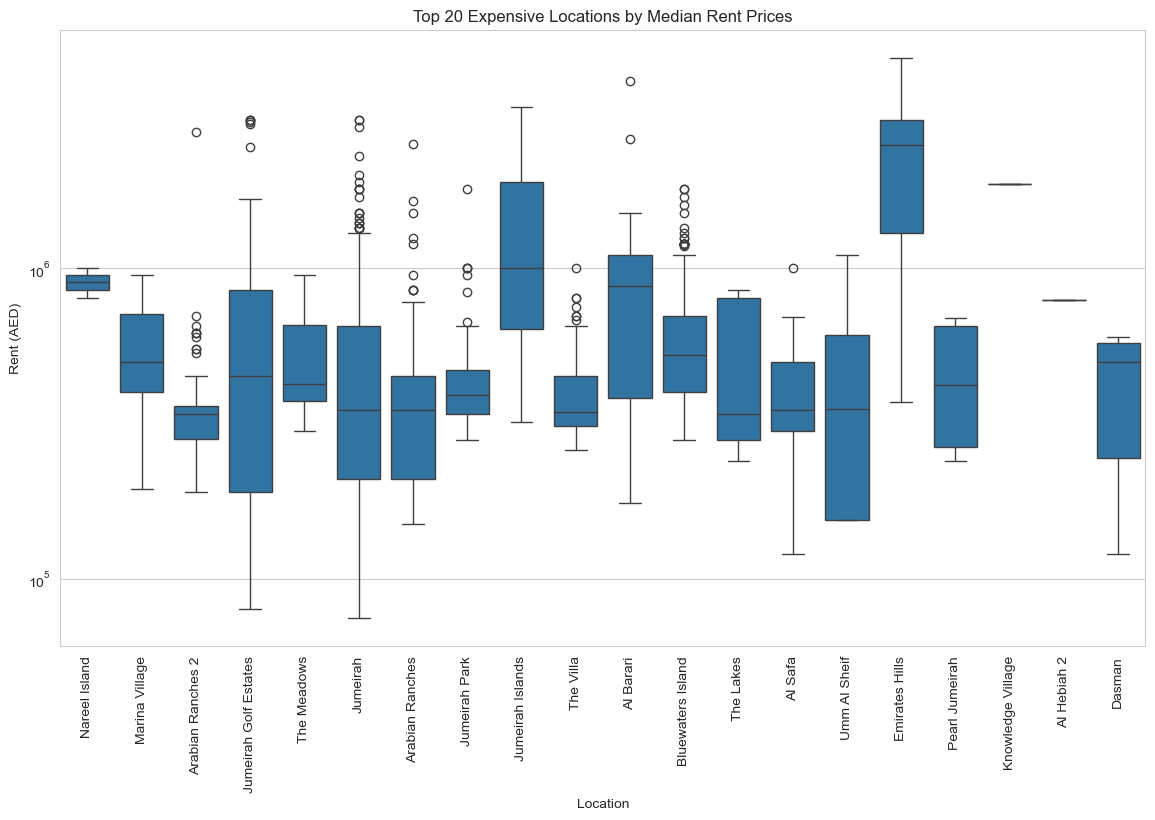


图 3-8-2租金中位数排名前20位的地点

为了进一步分析地理位置的房租租金需求分布情况，选取租金需求最高的前20个地区绘制柱状图，见图3-8-3。由图表可知Al Reem lsland 、Jumeirah Village Circle (JVC)、 Downtown Dubai 的房产数量最多，超过30000套，这些地区可能是迪拜的热门住宅区或商业区。

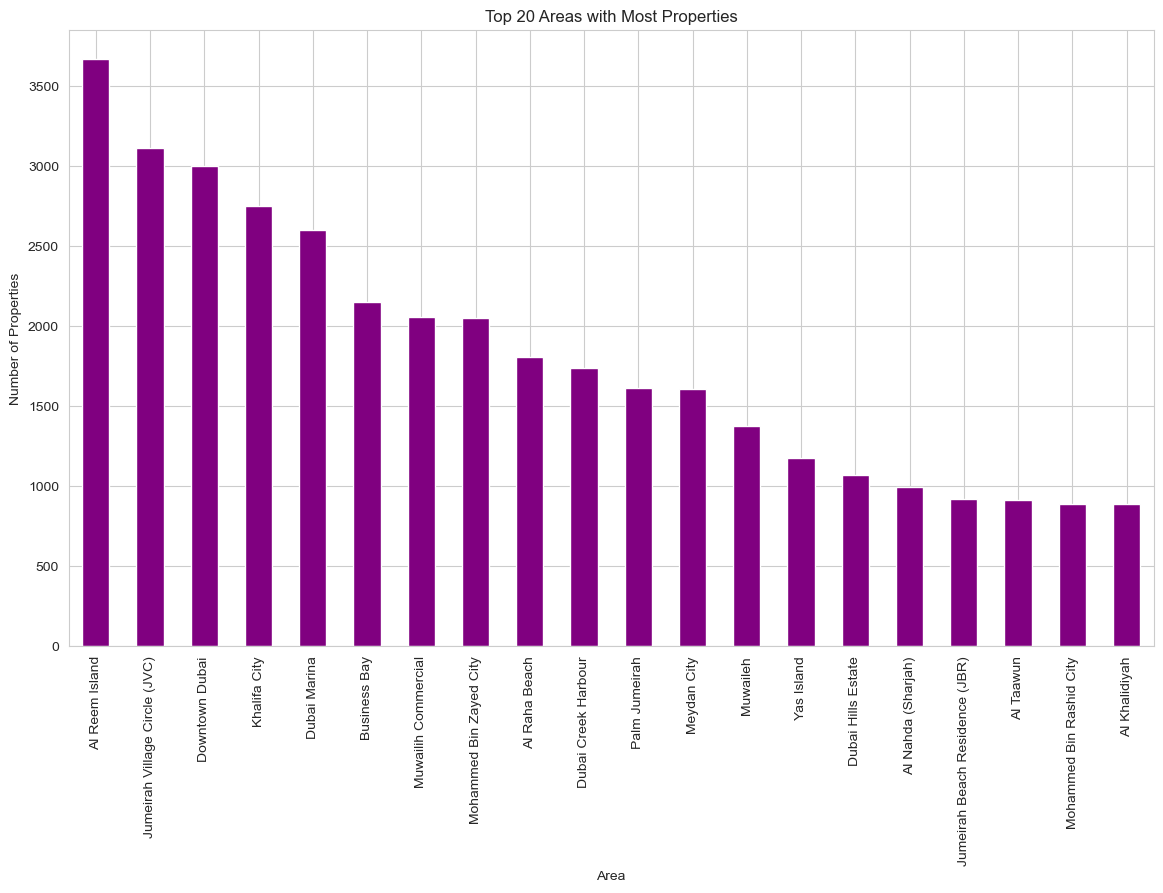


图 3-8-3租金需求最高的前20个地区绘制柱状图

## （九）城市对租金的影响

对城市与房价租金绘制箱线图，见图3-9-1，图表中租金价格的范围介于106 AED/m2到107 AED/m2，租金价格的波动范围较大；大多数城市的租金价格中位数集中在104 AED/m2到105 AED/m2之间，其中Dubai中位数最高，是租房市场比较火热的区域。在城市与租房需求的柱状图中，见图3-9-2，Dubai和Abu Dhabi租房需求远远超过其他城市的租房需求，其数量达20000以上，可能与城市的经济发展水平有关。

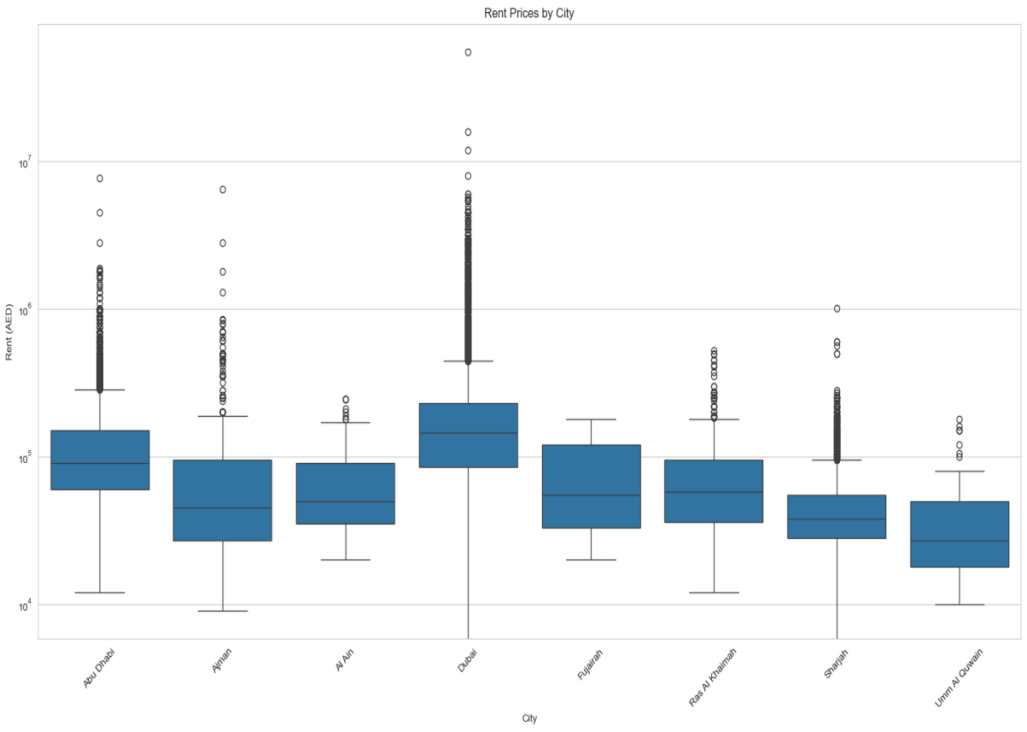


图 3-9-1城市与房价租金的箱线图

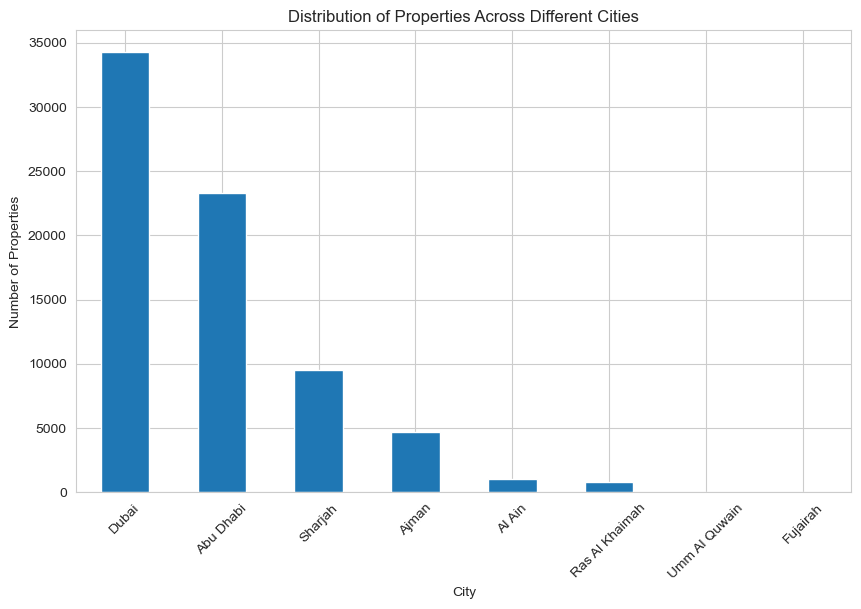


图 3-9-2城市与租房需求的柱状图

## （十）房产的家具状态对租金的影响

根据房产的家具状态分布图（图3-10）可知，没有家具配置的房产出租数量达54706套，有家具配置的房产出租数量达15617套，在家具状态与租金水平的对比图中，没有家具配置的租金水平均高于有家具配置的租金阈值，该现象表明消费群体在面临价格和物质需求选择上，侧重于物质需求，选择自己配置家具。

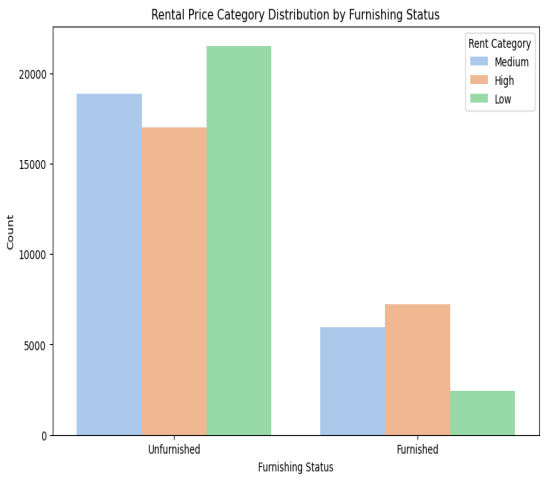
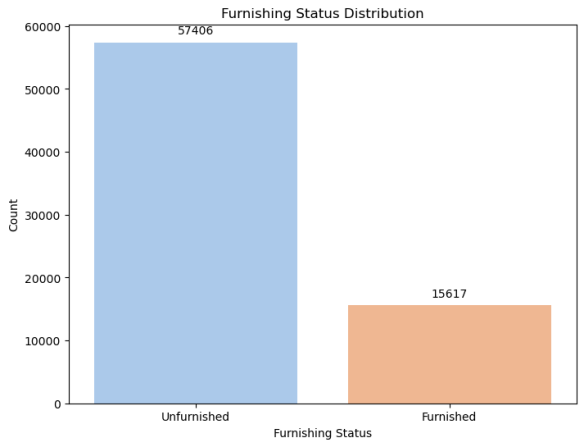


图 3-10家具状态与租金水平的对比图

## （十一）挂牌天数与出租需求分布图

通过挂牌天数分布图（图3-11）可以观察到，房屋在挂牌50天内的出租需求较高，在挂牌后100天内出租需求较小，可能是处于信息发达的时代，消费群体能不断获取最新的出租信息导致对挂牌天数久远的房屋出租信息被覆盖掉。

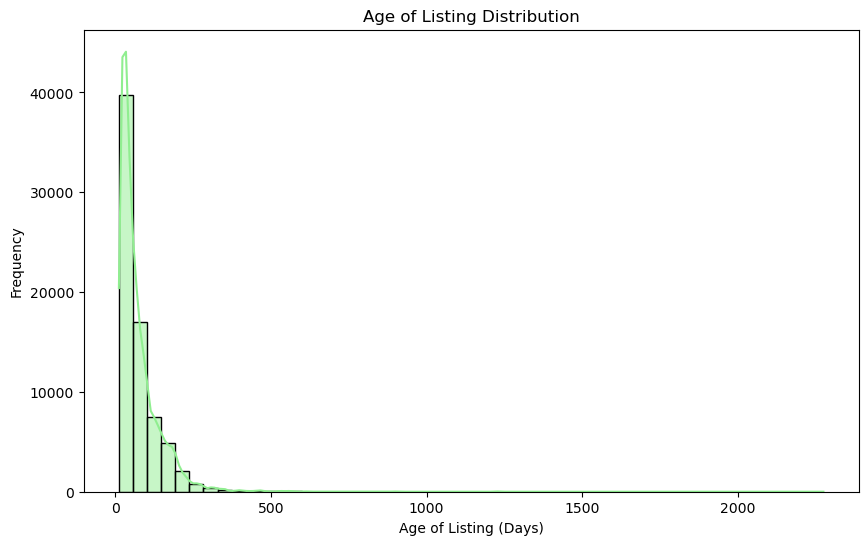


图 3-11挂牌天数分布

## （十二）房产的用途租金分布

在房产的用途租金箱线图（图3-12）中可以看出，租金用途主要用于出租，租金整体集中在0到106AED，但存在一些异常值，如高于5x107AED，可能与房租类型、地理位置等因素有关。



图 3-12房产用途与租金分布

## （十三）挂牌出租的日期对房价的影响

利用python将'Posted\_date'转换为datetime格式，绘制时间与平均租金折线图，见图3-13，由折线图可知，在2022年房价变化较大，可能受新冠疫情影响导致，疫情结束后后，在2023上半年，房产出租市场逐渐恢复。通过观察季节与平均租金的柱状图，房租在第二季度和第三季度的较低，特别是在五月到七月，可能与阿酉联所处位置有关，受气候影响居民可能选择去其他地方避暑。

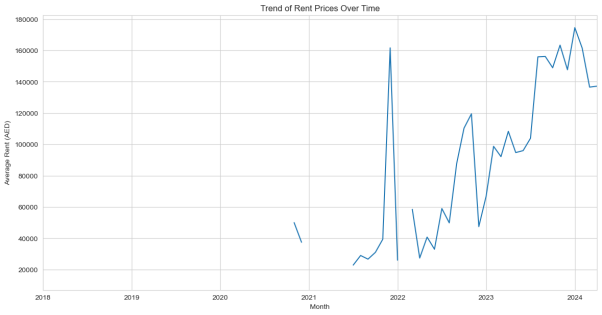


图 3-13挂牌出租日期与房价的分布

# 第四章 简单基本模型

在建立预测模型的过程中，当分类自变量的取值过多时，会导致模型的维度过大，自由度降低，增加了模型参数估计的难度，对样本质量要求很高。在本文使用的73742条数据的数据集中，我们对每行涉及缺失值或者空值进行删除，保留了73023 条数据。随机选取 80% 的数据作为训练集，20% 的数据作为测试集。

## （一）线性回归(Linear Regression)

线性回归是一种简单的统计方法，用于预测因变量的值基于一个或多个自变量的值。本文主要对房租进行预测，其特征主要有地址、卧室数量、浴室数量等，对此我们可以构建多元线性回归方程，其数学表达式为：

其中 表示第个样本向量 中第 维特征值。

## （二）决策树(Decision Tree)

决策树是一种自上而下，对样本数据进行树形分类的过程，由结点和有向边组成。结点分为内部结点和叶结点，其中每个内部结点表示一个特征或属性，叶节点表示类别。从顶部根节点开始，所有样本聚在一起。经过根节点的划分，样本被分到不同的子节点中。再根据子节点的特征进一步划分，直至所有样本都被归到某一类别（即叶节点）中。

在sklearn中用.fit()训练模型，它接受特征矩阵X\_train和目标值y\_train作为输入，并根据这些数据构建决策树。决策树是通过递归地分割数据空间来构建的，每次分割都是基于某个特征的条件，以最大化信息的增益或最小化杂质（如基尼不纯度或信息增益），具体公式如下：

1信息熵

样本集合为，信息熵的值越小，则的纯度越高。

2信息增益Gain(D,a)

给分支结点赋予权重样本数越多的分支结点的影响越大，信息增益越大，使用属性对样本集进行划分所所获得的纯度提升越大。

3增益率





属性的可能取值数目越多，则的值通常会越大。

4基尼指数



其中的值越小，则的纯度越高。

## （三）XGboost(（eXtreme Gradient Boosting）

XGBoost是一种流行的机器学习库，用于实现梯度提升决策树算法。XGBoost首先初始化一个弱学习器（通常是决策树），这个学习器的输出是所有预测值的平均值。在每一轮迭代中，XGBoost会计算当前模型对每个训练样本的残差，即实际目标值与当前模型预测值之间的差异，再使用这些残差作为目标变量来训练一个新的决策树。决策树通过选择最优的特征和切分点来最小化目标函数，公式如下：





## （四）模型预测结果

对房屋租赁信息的测试集使用交叉验证分数方法过程中，调用sklearn，分别对线性回归、决策树、XGboost模型预测结果，并使用以下三个指标对模型结果进行评价。其中代表真实值， 代表预测值，这三个指标均衡量了预测值与真实值之间的差异程度。

1.均方误差()



2.均方根误差()



3.拟合优度()



对三个模型预测结果进行对比，具体见表4-1，我们可以看到线性回归模型拟合效果最差，其预测误差MSE远高于决策树模型，说明存在较大的误差，难以准确反映数据的整体趋势；决策树模型模型具有较高的拟合优度，能够较好地捕捉数据的趋势，然而，其MSE相对较高，可能存在一定的误差；XGboost模型具有较好的拟合性能，能够有效地预测数据，虽然其MSE略高，处于可接受的范围内。

表4-1模型预测结果对比图

|  |  |  |  |
| --- | --- | --- | --- |
|  | MSE | RMSE | R2 |
| 决策树模型 | 3200994451.37 | 56577.33 | 0.96 |
| 线性回归模型 | 42904822281.70 | 207134.79 | 0.25 |
| XGboost模型 | 2965456044.99 | 54456.00 | 0.96 |

## （五）特征权重

鉴于三种模型的对比，初步在XGboost模型基础上使用scikit-learn中进行特征选择，可以得到每个属性的重要性得分，见图4-1。卧室数量 特征重要值排最高，值约为0.58，租金阈值、房产总面积、房产的单价特征重要值相对较高，值均在0.09以上，其他特征重要值相对较低，均在0.05以下。表明房产出租主要用于满足一些简单居住需求，消费群体看重房租的价格与面积，对所处位置、浴室数量需求较低。

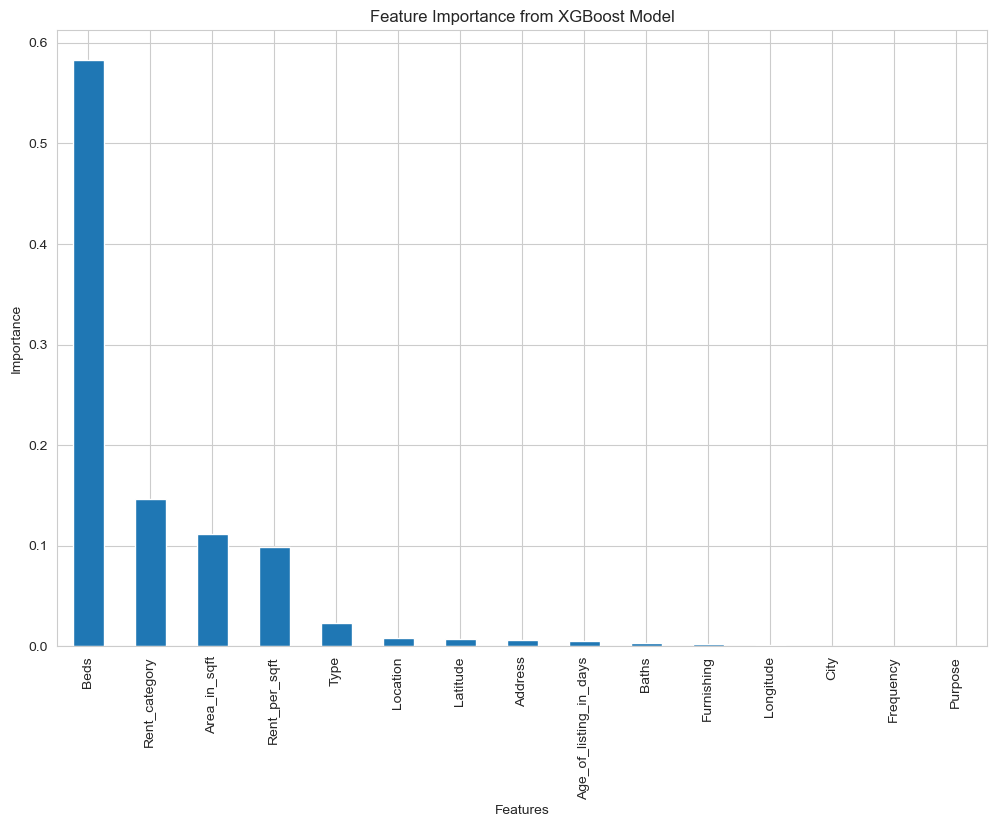


图 4-1 特征重要值排序

# 第五章 深度学习

鉴于上章节的三种模型对比，虽然XGboost能较好的拟合性能，但其MSE略高，显示不是最优的模型。对此，采用深度学习方法进一步探究对房租预测拟合最优的模型。

## （一）神经网络（Neural Networks，NN）

神经网络是一种模仿人脑工作方式的计算模型，它通过大量的节点（或称为“神经元”）和层来处理信息。神经网络通常包含多个隐含层，这些层之间的节点相互连接，形成复杂的网络结构。每个隐含层都能够从输入数据中学习到更高层次的特征表示，这些特征在下一层中被进一步抽象和转化。通过这种方式，神经网络能够断调整这些权重，以达到对数据进行分类、识别或预测等目的。

### 5.1.1特征工程

首先对数据清洗，删除所有含有至少一个NaN值的行，然后需要对数据进行标准化和归一化，使得数据的分布在同一范围内，再对类别特征进行编码，以便算法能够处理这些特征，具体公式如下：

1.数据标准化公式



这样，标准化后的特征将具有零均值和单位标准差，使得所有特征都在同一个量级上，有助于模型的训练和预测。

2.数据归一化公式



在处理时采取批量归一化，使用from sklearn.model\_selection import train\_test\_split 进行数据集划分，设置random\_state=42进行数据混洗， 将测试集与输入数据集比例设置为0.1，得到训练集样本量为65720，测试集样本量为7303，选取特征维度为16。

### 5.1.2模型的建立

定义一个多层感知机(MLP), 首先定义了五个全连接层（nn.Linear），分别有输入层到第一个隐藏层、第一个隐藏层到第二个隐藏层、…、第四个隐藏层到第五个隐藏层的连接。由于激活函数的选择通常依赖于具体的应用和任务，没有一种通用的最佳激活函数。因此，在设计和训练深度学习模型时，尝试不同的激活函数并选择性能最佳的那个。对此，将激活函数和进行对比，见图5-1。激活函数在时，梯度为0；在时，梯度为1，而激活函数在时梯度不为0，且在整个定义域内平滑变化。因此，我们选取作为MLP的激活函数，有助于保持训练过程中的梯度流动，这可能有助于提高模型的训练效率和性能，其中 激活函数表达式：

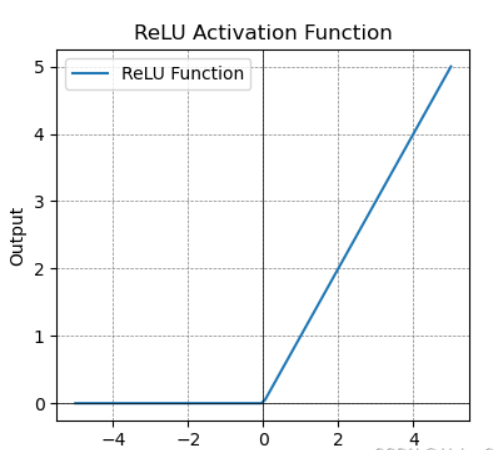
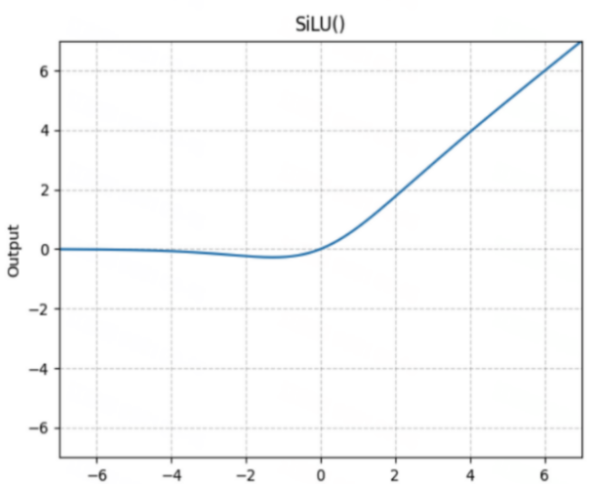


图5-1 SiLU激活函数图像（左）、ReLU激活函数图像（右）

接下来进入神经网络的训练阶段，主要分为以下两个阶段：

第一个阶段——前向传播，需要计算层所有神经元的加权输入，使用公式如下：

其中是连接第层的第个神经元和第层的第个神经元的权重，是第层的第个神经元的输出，是第层的第个神经元的偏置。

再使用激活函数输出神经元，公式如下：

第二个阶段——反向传播，通过计算网络输出与实际输出之间的误差，并将这个误差信号按照网络的结构反向传播回网络，以此来训练调整网络的参数（权重和偏置）。

对于输出层第L层的第个神经元，其误差计算公式如下：

其中,是损失函数，是激活函数的导数，是输出层第个神经元的加权输入。

对于隐藏层l的第个神经元，其误差公式如下：



其中是下一层的神经元数量，是连接当前层第个神经元和下一层第个神经元的权重。

对于连接第层的第个神经元和第个神经元的权重，其梯度计算公式如下：



对于第层的第个神经元的偏置，其梯度计算公式为：



最后，使用梯度下降或其他优化算法来更新网络的权重和偏置：





其中是学习率。

### 5.1.3模型训练与验证

由于GPU具有高度并行化的架构，能够提供高性能计算，具有优化的内存带宽，能支持专门的库和框架，采取图形显卡作为训练硬件，将数据集从内存送到显存中。数据加载器创建后，指定批量大小建立实例化模型，定义损失函数表达式：



其中就是网络层的待学习的参数，则控制正则项的大小。

下一步进行优化器的选择，由于结合了优化器的自适应学习率特性，可以根据每个参数的梯度的一阶矩估计和二阶矩估计来调整学习率，进行快速收敛。然后对优化器设置学习率初始值0.0001、两个超参数，便于后续AdamW算法中的动量计算。同时，设置权重衰退值0.01，将每个参数的权重值在每次迭代时将减少其原始值的0.01。最后用DataLoader中加载数据，定义一个较小的batch\_size，给训练提供更多的梯度更新机会以及定义训练循环次数epochs。

在验证时，利用matplotlib绘制在训练过程中的损失（Loss）、均方误差（MSE）、均方根误差（RMSE）以及拟合优度R2的曲线图，详见图5-2。从图5-3可知：随着训练次数（epoch）从0递增到100，训练损失（Loss)、均方误差（MSE）、均方根误差（RMSE）总体呈现振荡下降，MLP可能在前20轮就已经存在局部最优解，但其拟合优度R2在epoch之间的变化相对较小，表明模型在测试集上的表现相对稳定。与XGboost模型的MSE相比，显然经过深度学习研究后的神经网络模型的MSE显著变小，但拟合度R2相对XGboost变小，还需进一步深层探究房租预测的最优模型。

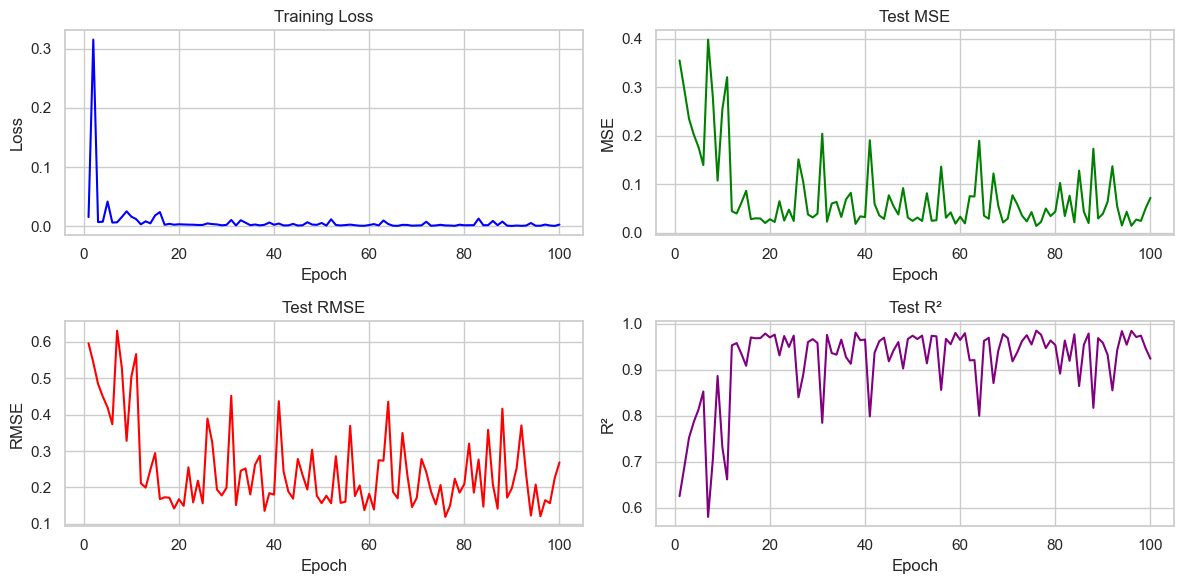


图5-2 Loss、MSE、RMSE、R2在训练过程中的曲线图

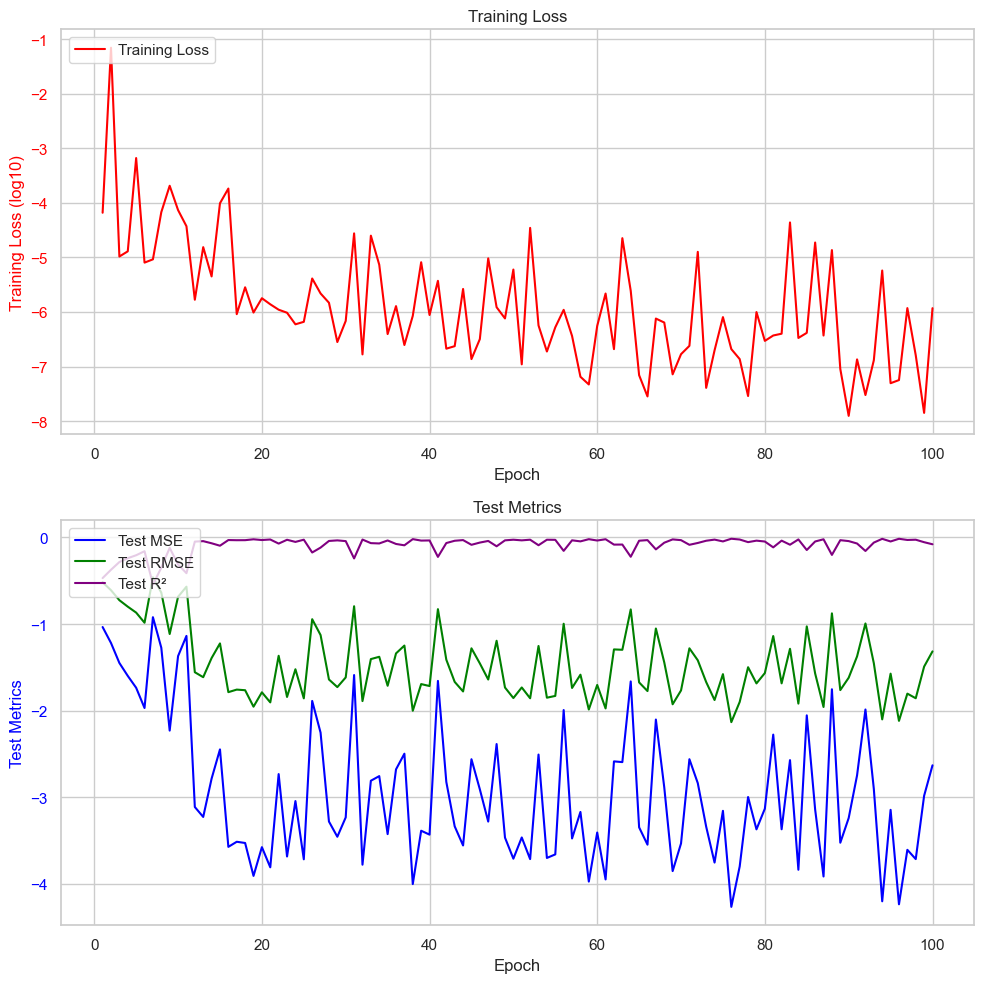


图 5-3四种参数重构的数据可视化曲线图

## （二）长短时记忆网络（Long Short-Term Memory，LSTM）

基于上述神经网络，进一步研究阿联酉房租预测的最优模型。这里引用LSTM模型，它是一种特殊的RNN（Recurrent Neural Network，循环神经网络）架构，通常被设计来解决传统RNN在处理长序列数据时遇到的梯度消失和梯度爆炸问题。LSTM通过引入“门”（gate）机制来实现对长期依赖信息的有效学习。

### 5.2.1特征工程

特征工程与构建MLP前一致，详见章节5.1.1。

### 5.2.2模型的建立

由于数据是二维的，这里需要在添加一个序列长度维度转化为三维，再对LSTM的隐藏状态和细胞状态初始化为0，其中是隐藏状态包含了LSTM单元在处理序列时的短期记忆，而细胞状态则包含了长期记忆。接下来对LSTM通过引入门结构来控制信息流入和流出细胞状态的过程。其中门结构主要有以下三种：

1.遗忘门（）： 遗忘门决定了哪些信息应该从细胞状态中丢弃。通过sigmoid的激活函数查看上一个隐藏状态（）和当前输入（），并输出一个介于0和1之间的值给每个在细胞状态（）中的数。1表示“完全保留这个信息”，而0表示“完全丢弃这个信息”，具体公式如下：



2输入门（）： 输入门负责更新细胞状态。首先sigmoid函数会决定哪些值将要更新，然后tanh函数创建一个新的候选值向量使得可以被加到状态中，具体公式如下：





3.输出门（）： 输出门决定时间步将输出什么值。在输出前需要查看上一个隐藏状态和当前输入，再通过sigmoid函数决定细胞状态的哪个部分将输出，具体公式如下：



其中，在信息流入和流出细胞状态过程中，遗忘门和输入门会结合生成细胞状态的更新（），具体公式如下：



同时，还会根据新的细胞状态来更新隐藏状态，隐藏状态的更新（ht）具体公式如下：



上述公式中，是权重矩阵，是偏置向量，是sigmoid函数，而tanh是双曲正切函数，方括号[ ]表示两个向量或矩阵的拼接，是输出门的输出。

最后，通过一个全连接层将LSTM的输出映射到最终的预测值，具体公式如下：



这里表示输出，和是全连接层的权重和偏置。

### 5.2.3模型训练与验证

模型训练过程与MLP一致，详见章节5.1.3。

在验证过程时，为了更具体对神经网络模型和LSTM模型做对比，使用平均绝对百分比误差（SMAPE）进行评估。

SMAPE不依赖于数据的尺度，因此在比较不同数据集或模型的性能时，它提供了一个统一的度量标准，公式如下：



在验证时，同样地利用matplotlib绘制在训练过程中的Loss、MSE、 RMSE以及SMAPE的曲线图，详见图5-4。从图5-5可知：随着训练次数（epoch）从0递增到100，训练损失波动下降，平均绝对百分比误差（SMAPE)波动也较下，其趋势整体在下降。而均方误差（MSE）、均方根误差（ RMSE）以及在epoch之间的变化相对较小，可以看出模型比神经网络在测试集上的表现更稳定。

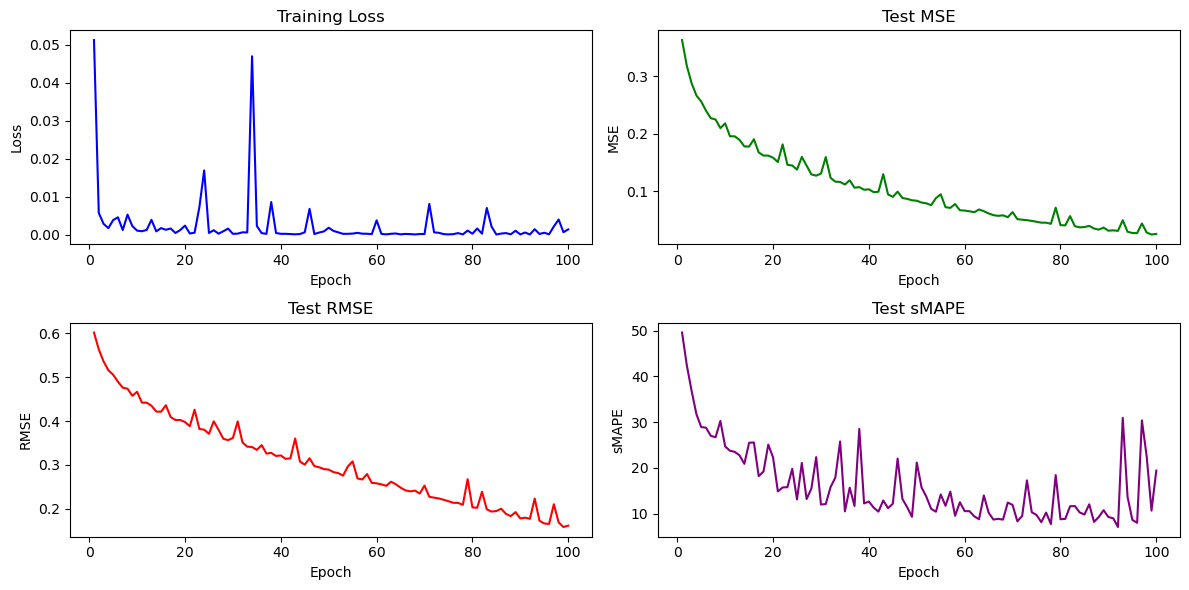


图 5-4 Loss、MSE、 RMSE以及SMAPE的曲线图

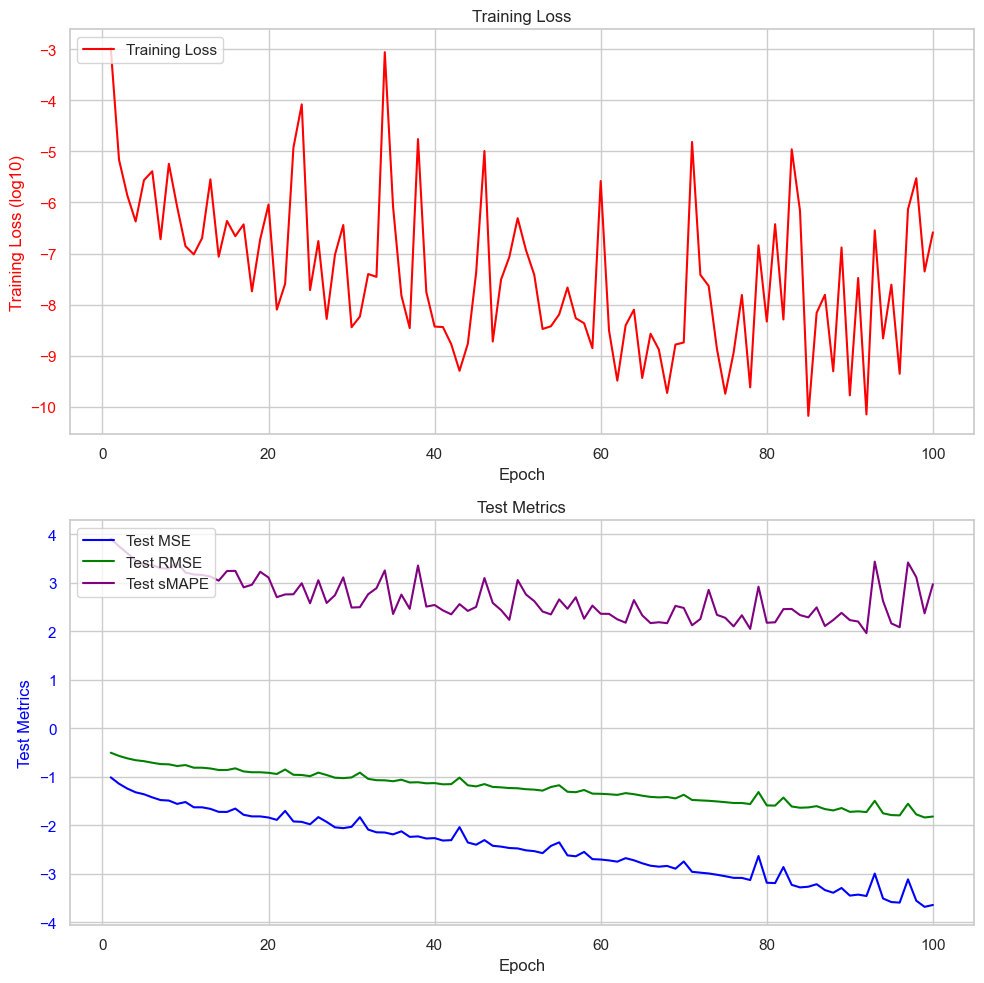


图 5-5 四种参数重构的数据可视化曲线图

现对两个模型的评估参数进行具体的对比，见表5-1，从表中可以看出深度学习在模型估测中都表现出较高的预测准确性。然而，LSTM 模型的整体表现略优于神经网络模型，尤其是在均方误差MSE和拟合优度R²这两个关键指标上。因此，考虑到精确度和可靠性，LSTM 模型是一个最佳的模型选择。

多次进行两个模型的训练对比，取较好的模型，参数详见: /pre\_models/MLP 和 /pre\_models/LSTM

表 5-1 神经网络模型与LSTM模型的评估参数对比

|  |  |  |
| --- | --- | --- |
|  | SMAPE | R2 |
| 神经网络模型 | 27.7 | 0.960 |
| LSTM模型 | 19.410 | 0.973 |

将多次训练结果保存到模型路径中，对比参数可得知，在batch\_size较小的情况下，两种神经网络模型的R方等指标有明显的提升，但也有过拟合风险，Adamw优化器在MLP的表现先相对较差，数量级范围内震荡幅度较大，不便于稳定收敛，LSTM在训练过程表现较好，偶有轮次会因为动量等因素突破数量级范围增加一次损失值。LSTM的sMAPE表现多比MLP好，R方与xgboost和MLP接近，说明拟合效果同样较好，且泛化能力更强，鲁棒性更强。训练过程中还会调节各种超参数改变模型训练效果和训练时间，本文重点只在简单训练不同的模型进行多维数据集并查看拟合效果，篇幅有限不再着重论述超参数对神经网络的各种影响。

# 第六章 总结

本文通过对阿联酋房地产市场房租数据的深入研究，采用机器学习和深度学习技术，成功构建了多个房租预测模型，并对模型结果进行了详尽的分析和解释，得到以下结论：

1.在预测阿联酋房租方LSTM模型展现出了卓越的性能，其预测误差最低，且拟合优度也相对较高，这一发现充分证明了LSTM模型在捕捉房租变化规律方面的有效性。

2.在特征重要性分析中，我们发现卧室数量是影响房租的首要因素，这反映了在租房市场上，消费者对于居住空间大小的强烈需求。其次，房产面积和每平方米租金等因素也对房租产生显著影响，这进一步验证了租房者在选择住房时，不仅关注居住空间的大小，还十分注重租金价格的合理性。

尽管本研究在阿联酋房地产市场的房租预测方面取得了显著的成果，但仍存在一些不足之处，这些不足为我们未来的研究提供了改进的方向。

1.样本量的大小和维度限制了研究的广度。由于获取完整而庞大的房地产数据可能存在技术和时间上的困难，我们的研究样本相对有限。这可能导致模型在泛化能力上有所欠缺，无法完全代表整个阿联酋房地产市场的房租变化规律。因此，在未来的研究中，需要努力扩大样本规模，以更全面地了解房租市场动态。

2.虽然已考虑了部分关键特征，但可能仍有其他重要的影响因素被遗漏。房地产市场是一个复杂的系统，涉及众多因素如地理位置、交通便利性、时间、周边设施等，这些因素都可能对房租产生显著影响。因此，在未来的研究中，我们需要进一步探索并引入更多与房租相关的特征，以构建更为全面和准确的预测模型。

# 致谢

1. Python: 作为一种高效的数据分析和机器学习编程语言，Python为本研究提供了强大的支持。我们特别感谢Python社区提供了丰富的开源库，使数据处理、模型建立和结果分析变得更加便捷。
2. scikit-learn: 作为Python中常用的机器学习库，scikit-learn为本研究提供了决策树、线性回归、XGBoost等多种模型。我们感谢scikit-learn社区的努力和贡献。
3. PyTorch: PyTorch是一个快速发展的开源深度学习框架，其灵活易用的特点为本研究中神经网络的实现提供了很大帮助。我们感谢PyTorch团队的工作。
4. matplotlib: matplotlib是一个强大的数据可视化库，为本研究中数据的可视化分析提供了便捷的支持。我们向matplotlib团队表示衷心的感谢。
5. bayut.com: 我们从bayut.com网站获取了阿联酋房地产市场的数据集，为本研究提供了宝贵的数据资源。我们向bayut.com表示衷心的感谢

# 参考文献

1. 高晨辉. 基于大数据的杭州市租房价格预测研究[J]. 商业时代, 2022(5): 2-3.
2. 夏静伶. 基于Python的武汉市住房租金预测研究[J]. 计算机工程与应用, 2022, 58(2): 122-123.
3. Engsted, T., & Pedersen, T. Q. (2015). Predicting returns and rent growth in the housing market using the rent-price ratio: Evidence from the OECD countries. Journal of International Money and Finance, 53, 257-275.
4. Khandaskar, S., Panjwani, C., Patil, V., Fernandes, D., & Bajaj, P. (2023). House and Rent Price Prediction System using Regression. In 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS).
5. Truong, Q., Nguyen, M., Dang, H., & Mei, B. (2020). Housing Price Prediction via Improved Machine Learning Techniques. Procedia Computer Science, 174, 433-442.
6. Xu, X., & Zhang, Y. (2021). House price forecasting with neural networks. Intelligent Systems with Applications, 12, 200052.

# 附录

基本数据分析与可视化过程: /analysis\_main.ipynb

模型/pre\_models/\*

机器学习/analysis\_ML.ipynb

MLP/analysis\_DL.ipynb

LSTM/analysis\_LSTM.ipynb

由于代码量较大，只展示部分内容

1.神经网络/analysis\_ML.ipynb

import pandas as pd

import numpy as np

data=pd.**read\_csv**("dubai\_properties.csv")

data.**dropna**(inplace=True)

**print**(data.**head**(3))

# data

# 查看数据类型

**print**(data.dtypes)

from sklearn.preprocessing import LabelEncoder

## 对非数值类型的特征进行分类编码

# 创建一个字典来存储每个特征的编码器

encoders = {}

# 遍历每列

for column in data.columns:

    # 如果是对象类型（通常是字符串或混合类型）

    if data[column].dtype == object:

        # 创建一个 LabelEncoder

        encoder = LabelEncoder()

        # 对当前列进行编码

        data[column] = encoder.**fit\_transform**(data[column])

        # 保存编码器以便后续使用

        encoders[column] = encoder

    elif data[column].dtype == 'int64' or data[column].dtype == 'float64':

        # 数值类型，不需要编码

        pass

    else:

**print**(f"Warning: Unhandled data type for column '{column}'")

# 查看数据

**print**(data.**head**(2))

**print**(data.dtypes)

# 将编码后的数据保存到CSV文件，便于其他程序调用

encoded\_data\_file = 'encoded\_dataset.csv'

data.**to\_csv**(encoded\_data\_file, index=False)

**print**(f"Encoded data saved to {encoded\_data\_file}")

# 准备特征和目标变量

X = data.**drop**(['Rent'],  axis=1)

y = data['Rent']

# # 根据分类剔除分类为1的维度

# X = X.drop(['Frequency', 'Purpose'], axis=1)

**print**(X.**head**(2))

**print**(y.**head**(2))

from sklearn.preprocessing import StandardScaler

# 使用标准归一化器

standard\_scaler = StandardScaler()

X\_scaled = standard\_scaler.**fit\_transform**(X)

**print**(X\_scaled[:2])

**print**(X\_scaled.shape)

**print**(y.shape)

# 归一化正确，将y也归一化处理

y\_from\_to\_frame = y.**to\_frame**()

**print**(y\_from\_to\_frame.shape)

y\_scaled = standard\_scaler.**fit\_transform**(y\_from\_to\_frame)

**print**(y\_scaled[:2])

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.model\_selection import **train\_test\_split**

# 划分数据集

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_scaled, y\_scaled, test\_size=0.1, random\_state=42) # 批量归一化

X\_train\_tensor = torch.FloatTensor(X\_train)

y\_train\_tensor = torch.FloatTensor(y\_train)

X\_test\_tensor = torch.FloatTensor(X\_test)

y\_test\_tensor = torch.FloatTensor(y\_test)

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42) # 非批量归一化

# X\_train\_tensor = torch.FloatTensor(X\_train.values)

# y\_train\_tensor = torch.FloatTensor(y\_train.values)

# X\_test\_tensor = torch.FloatTensor(X\_test.values)

# y\_test\_tensor = torch.FloatTensor(y\_test.values)

**print**(X\_train\_tensor[:3].numpy())

**print**("训练集特征张量大小", X\_train\_tensor.size())

**print**("训练集目标张量大小", y\_train\_tensor.size())

**print**("测试集特征张量大小:", X\_test\_tensor.size())

**print**("测试集目标张量大小:", y\_test\_tensor.size())

# 构建神经网络

class RentPredictor(nn.Module):

    def **\_\_init\_\_**(self, input\_dim):

        super(RentPredictor, self).\_\_init\_\_()

        self.layer1 = nn.Linear(input\_dim, 64)

        self.layer2 = nn.Linear(64, 128)

        self.layer3 = nn.Linear(128, 256)

        self.layer4 = nn.Linear(256, 64)

        self.layer5 = nn.Linear(64, 1)

        self.swish = nn.SiLU()

    def **forward**(self, x):

        x = self.swish(self.layer1(x))

        x = self.swish(self.layer2(x))

        x = self.swish(self.layer3(x))

        x = self.swish(self.layer4(x))

        x = self.layer5(x)

        return x

from torch.utils.data import DataLoader

# 将模型放到GPU上运行

if torch.cuda.is\_available():

    device = torch.device("cuda")

**print**(f"Using GPU: {torch.cuda.get\_device\_name(0)}")

else:

    device = torch.device("cpu")

**print**("Using CPU")

# 将数据也送到GPU

X\_train\_tensor\_gpu = X\_train\_tensor.to(device)

X\_test\_tensor\_gpu = X\_test\_tensor.to(device)

y\_train\_tensor\_gpu = y\_train\_tensor.to(device)

y\_test\_tensor\_gpu = y\_test\_tensor.to(device)

# 创建数据加载器，指定批量大小

batch\_size = 128

train\_dataset = torch.utils.data.TensorDataset(X\_train\_tensor\_gpu, y\_train\_tensor\_gpu)

test\_dataset = torch.utils.data.TensorDataset(X\_test\_tensor\_gpu, y\_test\_tensor\_gpu)

train\_data\_loader = DataLoader(dataset=train\_dataset, batch\_size=batch\_size, shuffle=True)

test\_data\_loader = DataLoader(dataset=test\_dataset, batch\_size=batch\_size, shuffle=False)  # 关闭打乱

# 实例化模型

**print**(f"shape X\_train\_tensor\_gpu: {X\_train\_tensor\_gpu.shape}")

# print(f"shape[0] y\_test\_tensor\_gpu: {y\_test\_tensor\_gpu.shape[0]}")

model = RentPredictor(X\_train\_tensor.shape[1]).to(device)

# 定义损失函数

criterion = nn.MSELoss()

# criterion = nn.SmoothL1Loss()

# 定义优化器

# optimizer = optim.SGD(model.parameters(), lr=0.0001, momentum=0.9)

# optimizer = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.999))

optimizer = optim.AdamW(model.parameters(), lr=0.0001, betas=(0.9, 0.999), weight\_decay=0.01)

# optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99)

# optimizer = optim.Adagrad(model.parameters(), lr=0.01)

# optimizer = optim.NAdam(model.parameters(), lr=0.002, betas=(0.9, 0.999))

# 定义训练轮数

epochs = 100

from sklearn.metrics import **mean\_squared\_error**, **r2\_score**

# 存储训练log

loss\_values = []

mse\_values = []

rmse\_values = []

r2\_values = []

# 训练模型

for epoch in range(epochs):

    # 训练模式

    model.train()

    for batch in train\_data\_loader:

        X\_batch, y\_batch = batch

        optimizer.zero\_grad()

        # 前向传播

        y\_pred = model(X\_batch)

        loss = criterion(y\_pred, y\_batch)

        # 反向传播

        loss.backward()

        optimizer.step()

    loss\_values.**append**(loss.item())

    if (epoch + 1) % 5 == 0:

**print**(f'Epoch {epoch+1}/{epochs}, Loss: {loss.item()}')

    # 评估模式

    model.eval()

    with torch.no\_grad():

        y\_val\_pred = model(X\_test\_tensor.to(device))

        # 将预测结果从GPU复制到CPU

        y\_val\_pred\_cpu = y\_val\_pred.to('cpu')

        # 将预测结果转换为Numpy数组

        y\_val\_pred\_numpy = y\_val\_pred\_cpu.numpy()

        # 安全地使用y\_val\_pred\_numpy作为mean\_squared\_error的输入

        mse\_lr = **mean\_squared\_error**(y\_test, y\_val\_pred\_numpy)

        rmse\_lr = np.sqrt(mse\_lr)

        r2\_lr = **r2\_score**(y\_test, y\_val\_pred\_numpy)

        mse\_values.**append**(mse\_lr)

        rmse\_values.**append**(rmse\_lr)

        r2\_values.**append**(r2\_lr)

import matplotlib.pyplot as plt

import seaborn as sns

# 绘制2x2的图表矩阵

plt.**figure**(figsize=(12, 6))

# 绘制训练损失曲线

ax1 = plt.**subplot**(2, 2, 1)

plt.**plot**(range(1, epochs + 1), loss\_values, label='Training Loss', color='blue')

plt.**title**('Training Loss')

plt.**xlabel**('Epoch')

plt.**ylabel**('Loss')

# 绘制MSE曲线

ax2 = plt.**subplot**(2, 2, 2)

plt.**plot**(range(1, epochs + 1), mse\_values, label='Test MSE', color='green')

plt.**title**('Test MSE')

plt.**xlabel**('Epoch')

plt.**ylabel**('MSE')

# 绘制RMSE曲线

ax3 = plt.**subplot**(2, 2, 3)

plt.**plot**(range(1, epochs + 1), rmse\_values, label='Test RMSE', color='red')

plt.**title**('Test RMSE')

plt.**xlabel**('Epoch')

plt.**ylabel**('RMSE')

# 绘制R²曲线

ax4 = plt.**subplot**(2, 2, 4)

plt.**plot**(range(1, epochs + 1), r2\_values, label='Test R²', color='purple')

plt.**title**('Test R²')

plt.**xlabel**('Epoch')

plt.**ylabel**('R²')

# 调整布局并显示图表

plt.**tight\_layout**()

plt.**show**()

# 设置seaborn样式

sns.**set**(style="whitegrid")

# 创建一个图和轴用于绘制训练损失

# viridis:高对比度的颜色方案

fig, ax1 = plt.**subplots**(figsize=(10, 5))

ax1.**set\_xlabel**('Epoch')

ax1.**set\_ylabel**('Training Loss', color=sns.**color\_palette**("viridis", 4)[0])

ax1.**plot**(range(1, epochs + 1), loss\_values, label='Training Loss', color=sns.**color\_palette**("viridis", 4)[0])

ax1.**tick\_params**(axis='y', labelcolor=sns.**color\_palette**("viridis", 4)[0])

ax1.**legend**(loc='upper left')

# 创建一个新的轴用于绘制其他指标，共享x轴

ax2 = ax1.**twinx**()

ax2.**set\_ylabel**('Test Metrics', color=sns.**color\_palette**("viridis", 4)[1])

ax2.**plot**(range(1, epochs + 1), mse\_values, label='Test MSE', color=sns.**color\_palette**("viridis", 4)[1])

ax2.**plot**(range(1, epochs + 1), rmse\_values, label='Test RMSE', color=sns.**color\_palette**("viridis", 4)[2])

ax2.**plot**(range(1, epochs + 1), r2\_values, label='Test R²', color=sns.**color\_palette**("viridis", 4)[3])

ax2.**tick\_params**(axis='y', labelcolor=sns.**color\_palette**("viridis", 4)[1])

ax2.**legend**(loc='upper right')

# 添加标题

plt.**title**('Training Loss and Test Metrics')

# 显示图表

plt.**show**()

# 重绘可视化数据

# 设置seaborn样式

sns.**set**(style="whitegrid")

# 创建一个图和两个轴用于绘制训练损失和其他指标

fig, (ax1, ax2) = plt.**subplots**(nrows=2, ncols=1, figsize=(10, 10))

# 对数变换

log\_loss\_values = np.log(loss\_values)

# 绘制训练损失曲线

ax1.set\_xlabel('Epoch')

ax1.set\_ylabel('Training Loss (log10)', color='red')

ax1.plot(range(1, **len**(log\_loss\_values) + 1), log\_loss\_values, label='Training Loss', color='red')

ax1.tick\_params(axis='y', labelcolor='red')

ax1.legend(loc='upper left')

# 对数变换

log\_mse\_values = np.log(mse\_values)

log\_rmse\_values = np.log(rmse\_values)

log\_r2\_values = np.log(r2\_values)

# 绘制MSE曲线

ax2.set\_xlabel('Epoch')

ax2.set\_ylabel('Test Metrics', color='blue')

ax2.plot(range(1, epochs + 1), log\_mse\_values, label='Test MSE', color='blue')

ax2.plot(range(1, epochs + 1), log\_rmse\_values, label='Test RMSE', color='green')

ax2.plot(range(1, epochs + 1), log\_r2\_values, label='Test R²', color='purple')

ax2.tick\_params(axis='y', labelcolor='blue')

ax2.legend(loc='upper left')

# 添加标题

ax1.set\_title('Training Loss')

ax2.set\_title('Test Metrics')

# 显示图表

plt.**tight\_layout**()

plt.**show**()

# 5. 计算指标

model.eval()

with torch.no\_grad():

    y\_test\_pred = model(X\_test\_tensor\_gpu)

    # print(f"shape: {y\_test\_pred.shape}")

    test\_loss = criterion(y\_test\_pred, y\_test\_tensor\_gpu.unsqueeze(1))

    # print(f"shape: {y\_test\_tensor\_gpu.shape}")

    mse = test\_loss.item()

    rmse = torch.sqrt(test\_loss).item()

# 打印指标

**print**(f'Test MSE: {mse}')

**print**(f'Test RMSE: {rmse}')

model.eval()

with torch.no\_grad():

    y\_val\_pred = model(X\_test\_tensor.to(device))

# 将预测结果从GPU复制到CPU

y\_val\_pred\_cpu = y\_val\_pred.to('cpu')

# 将预测结果转换为Numpy数组

y\_val\_pred\_numpy = y\_val\_pred\_cpu.numpy()

# 安全地使用y\_val\_pred\_numpy作为mean\_squared\_error的输入

mse\_lr = **mean\_squared\_error**(y\_test, y\_val\_pred\_numpy)

rmse\_lr = np.sqrt(mse\_lr)

r2\_lr = **r2\_score**(y\_test, y\_val\_pred\_numpy)

# 计算sMAPE

abs\_diff = np.abs(y\_test - y\_val\_pred\_numpy)

avg\_abs = (np.abs(y\_test) + np.abs(y\_val\_pred\_numpy)) / 2

sMAPE = np.mean((abs\_diff / avg\_abs) \* 100)

# 打印指标

**print**(f'Test MSE: {mse\_lr}')

**print**(f'Test RMSE: {rmse\_lr}')

**print**(f'Test R²: {r2\_lr}')

**print**(f'Test sMAPE: {sMAPE}')

# 查看模型参数

for name, param in model.named\_parameters():

**print**(f"{name}:", param.size())

# 选择测试集中的几个样本

X\_test\_sample = X\_test\_tensor\_gpu[:5]  # 选择前5个样本

y\_test\_sample = y\_test\_tensor\_gpu[:5]

# 使用模型进行预测

with torch.no\_grad():  # 确保不计算梯度

    y\_pred\_sample = model(X\_test\_sample)

# 将预测结果和实际目标值转换为 NumPy 数组

y\_pred\_sample\_np = y\_pred\_sample.cpu().numpy()

y\_test\_sample\_np = y\_test\_sample.cpu().numpy()

# 反归一化

y\_pred\_original = standard\_scaler.**inverse\_transform**(y\_pred\_sample\_np)

y\_test\_original = standard\_scaler.**inverse\_transform**(y\_test\_sample\_np)

# 打印反归一化后的预测结果和实际目标值

**print**('反归一化后的预测结果:', y\_pred\_original)

**print**('反归一化后的实际目标值:', y\_test\_original)

# 获取模型的所有参数

parameters = list(model.parameters())

# 将所有参数打包成一个向量

parameters\_vector = torch.cat([p.view(-1) for p in parameters])

# 计算参数量

num\_parameters = parameters\_vector.numel()

**print**(f"The model has {num\_parameters} parameters.")

import datetime

# 保存模型

# 获取当前日期和时间

current\_time = datetime.datetime.**now**().**strftime**('%Y%m%d\_%H%M%S')

# 结合模型名称和日期时间

model\_name = 'MLP\_Standard\_Adamw'

model\_file\_name = f"pre\_models/MLP/{model\_name}\_R{r2\_lr:.3f}\_B{batch\_size}\_S{sMAPE:.1f}.pth"

# 保存模型

torch.save(model, model\_file\_name)

**print**(f"Model saved as {model\_file\_name}")

2.LSTM代码 /analysis\_LSTM.ipynb, 某些过程和MLP相同，只展示网络模型等核心代码段，详见文件/analysis\_LSTM.ipynb

# 构建LSTM

class RentPredictor\_LSTM(nn.Module):

    def **\_\_init\_\_**(self, input\_dim, hidden\_dim, num\_layers, output\_dim):

        super(RentPredictor\_LSTM, self).\_\_init\_\_()

        self.hidden\_size = hidden\_dim

        self.num\_layers = num\_layers

        self.lstm = nn.LSTM(input\_dim, hidden\_dim, num\_layers, batch\_first=True)

        self.fc = nn.Linear(hidden\_dim, output\_dim)

    def **forward**(self, x):

        # x 的形状是 (batch\_size, feature\_dim)

        # 由于我们的数据是二维的，我们需要添加一个序列长度维度

        x = x.unsqueeze(1)  # 现在形状变为 (batch\_size, 1, feature\_dim)

        # 初始化隐藏状态和细胞状态

        h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)

        c0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)

        # 前向传播LSTM

        out, \_ = self.lstm(x, (h0, c0))

        # 解码最后一个时间步的隐藏状态

        out = self.fc(out[:, -1, :])

        return out

# 定义LSTM模型的参数

hidden\_dim = 128  # 隐藏层的维度

num\_layers = 4   # LSTM层的数量

output\_dim = 1  # 输出的维度

**print**(X\_train\_tensor.shape[1])

# 实例化模型

model = RentPredictor\_LSTM(input\_dim=X\_train\_tensor.shape[1],

                           hidden\_dim=hidden\_dim, output\_dim=output\_dim, num\_layers=num\_layers)

model.to(device)

# 定义损失函数

criterion = nn.MSELoss()

# 定义优化器

optimizer = optim.AdamW(model.parameters(), lr=0.0001, betas=(0.9, 0.999), weight\_decay=0.01)

# 定义训练轮数

epochs = 100

from sklearn.metrics import **mean\_squared\_error**, **r2\_score**

# 存储训练log

loss\_values = []

MSE\_values = []

RMSE\_values = []

sMAPE\_values = []

# 训练模型

for epoch in range(epochs):

    # 训练模式

    model.train()

    for batch in train\_data\_loader:

        X\_batch, y\_batch = batch

        optimizer.zero\_grad()

        # 前向传播

        y\_pred = model(X\_batch)

        loss = criterion(y\_pred, y\_batch)

        # 反向传播

        loss.backward()

        optimizer.step()

    loss\_values.**append**(loss.item())

    if (epoch + 1) % 5 == 0:

**print**(f'Epoch {epoch+1}/{epochs}, Loss: {loss.item()}')

    # 评估模式

    model.eval()

    with torch.no\_grad():

        y\_val\_pred = model(X\_test\_tensor.to(device))

        # 将预测结果从GPU复制到CPU

        y\_val\_pred\_cpu = y\_val\_pred.to('cpu')

        # 将预测结果转换为Numpy数组

        y\_val\_pred\_numpy = y\_val\_pred\_cpu.numpy()

        # 安全地使用y\_val\_pred\_numpy作为mean\_squared\_error的输入

        MSE\_lr = **mean\_squared\_error**(y\_test, y\_val\_pred\_numpy)

        RMSE\_lr = np.sqrt(MSE\_lr)

        # 计算sMAPE

        abs\_diff = np.abs(y\_test - y\_val\_pred\_numpy)

        avg\_abs = (np.abs(y\_test) + np.abs(y\_val\_pred\_numpy)) / 2

        sMAPE = np.mean((abs\_diff / avg\_abs) \* 100)

        MSE\_values.**append**(MSE\_lr)

        RMSE\_values.**append**(RMSE\_lr)

        sMAPE\_values.**append**(sMAPE)