

|  |  |
| --- | --- |
| **实验名称** | **实验二 聚类分析** |
| **姓 名** | **李灵慧** |
| **学 号** | **2202501-18** |
| **学 院** | **市政与测绘工程学院** |
| **专 业** | **地理空间信息工程** |
| **指导教师** | **汤淼** |

**2025年 3 月 20 日**

1 数据预处理

1.1数据问题

（1）大量空值

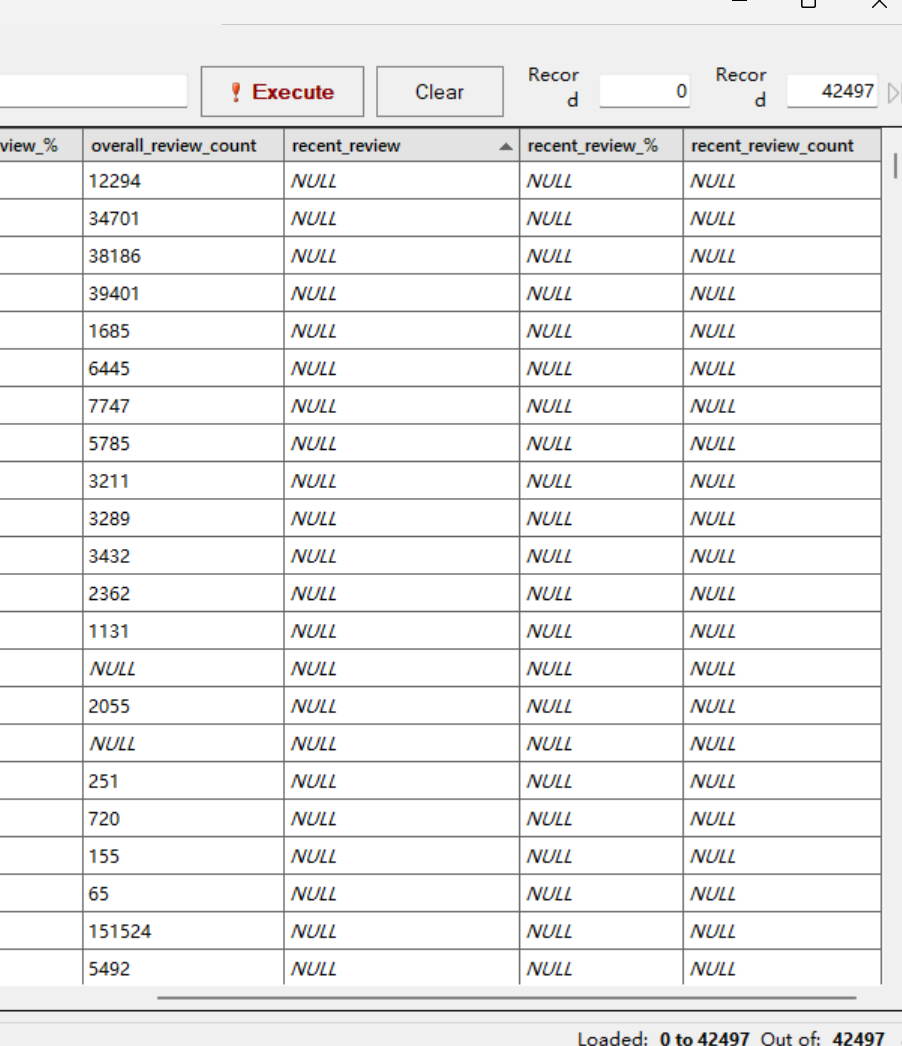


图 1.1.1 空值

（2）数值中带有字符，不利于计算和统计



图 1.1.2 数值带字符

（3）大量意义不明的字段，占用空间多

（4）字段值差异过大，不利于聚类分析，下面是价格和评论数量的值分布图，x轴是按大小排列后的数组序列，y轴是数组值，可以见到字段值变化呈现极端的“L”型，大多数元素的值集中在最大值的千分之一或万分之一以下。

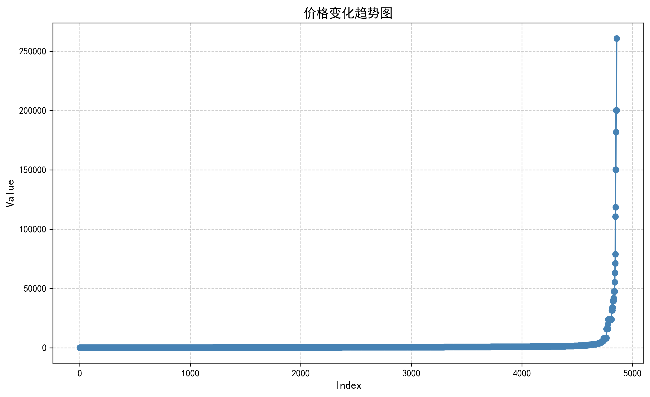
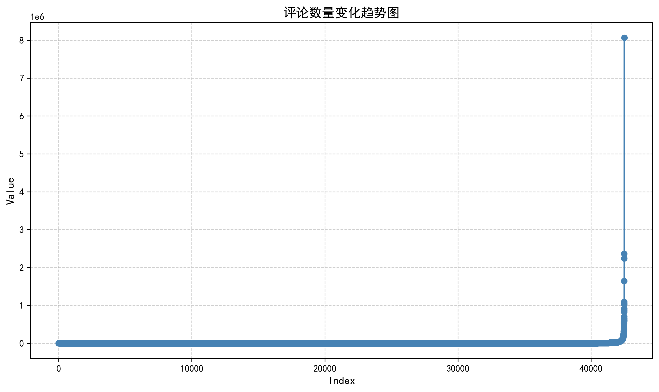
 

图1.1.3 值分布图

1.2应对方法

（1）填充空值为合适值，比如0

（2）使用正则表达式填充空值

（3）删去多余字段，保留"app\_id"、title"、"original\_price"、"overall\_review\_%"、"overall\_review\_count"五个用的着的字段。

（4）使用指数对数据尺度进行压缩，减少因数据差值过大造成的影响，下面是经过压缩后的效果，可以见到字段值的差异逐渐变小了。

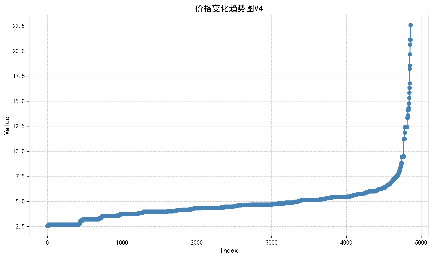
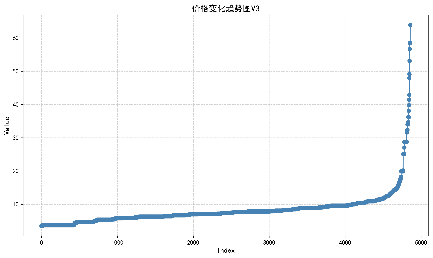
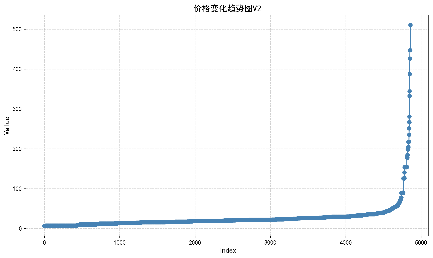


图1.2.1 价格字段值压缩

2聚类分析方法

2.1 直接分割法

介绍：根据直接输入的分割值分割数据

意义：分割值可以借鉴于生活常识或专业知识，分类结果不会很反常，主要是可以和另外两个方法形成对照，譬如对于游戏价格来说，小于1000卢比算便宜，1000-4000是一般游戏大作的定价，大于4000就是可以说是昂贵了。

直接分割法适合于有先验经验的数据，像是本次实验数据字段值差异大、无法直接从距离角度分类数据来说，是十分简单有效的分类方法。

**def** **SliptClusters**(data, StrID, StrCluster, SplitValues, label\_str):  
 """  
 根据分割值分类目标字段并打标签  
  
 参数：  
 data - 原始数据集（DataFrame）  
 StrID - 唯一标识字段名（字符串）  
 StrCluster - 需要分类的目标字段名（字符串）  
 SplitValues - 分割值数组（排序后的数值列表）  
 label\_str - 分类标签数组（长度需为len(SplitValues)+1）  
  
 返回：  
 包含StrID和分类结果的DataFrame  
 """  
 # 生成区间边界  
 bins = [-np.inf] + sorted(SplitValues) + [np.inf]  
  
 # 生成区间描述字符串  
 interval\_strings = []  
 **for** i **in** range(len(bins) - 1):  
 left = bins[i]  
 right = bins[i + 1]  
 # 处理无穷大显示  
 left\_str = '-∞' **if** left == -np.inf **else** f'{left:.2f}'  
 right\_str = '+∞' **if** right == np.inf **else** f'{right:.2f}'  
 interval\_str = f'[{left\_str}, {right\_str})' # 左闭右开  
 interval\_strings.append(interval\_str)  
  
 # 分箱操作  
 data['category'] = pd.cut(data[StrCluster], bins=bins, labels=label\_str, right=False)  
  
 # 按标签顺序统计（确保顺序与label\_str一致）  
 counts = data['category'].value\_counts().reindex(label\_str, fill\_value=0)  
  
 # 统计输出  
 print("\n\n区间分布统计：")  
 **for** label, interval, count **in** zip(label\_str, interval\_strings, counts):  
 print(f"{label}（{interval}）: {count}个")  
  
 # 返回指定字段  
 **return** data[[StrID, 'category']].rename(columns={'category': StrCluster+'\_judge'})

2.2 K-means

介绍：根据样本之间的距离进行类别分割，认为距离越近、相关性越高。

意义：K-means聚类是一种高效的无监督学习算法，通过迭代优化将数据划分为相似簇，核心意义在于简化数据复杂度并揭示潜在模式。其优势在于计算高效、易于实现，适用于多维场景，如客户分群、图像压缩和异常检测，成为数据探索与特征工程的实用工具。作为无监督学习的基石，K-means展示了迭代优化思想，并催生了轮廓系数等评估方法。然而，其局限性也推动算法发展：需预设簇数、对异常值敏感、假设簇呈球形分布，这些缺陷促使K-medians、DBSCAN等改进算法的诞生。实践中，K-means既能快速验证数据假设，又可通过簇标签辅助监督学习，平衡了效率与实用性。尽管存在不足，它仍是数据科学的核心工具，体现了“简化抽象”与“现实需求”的平衡，为后续复杂方法奠定基础，在业务分析和技术演进中持续发挥价值。

在本次实验中，对于未经过尺度压缩的样本，Kmeans的分类结果并不理想，但经过指数压缩变换后，也可以达到预期效果。

**def** **KMCluster**(data, StrID, StrCluster, Knum, label\_str=None, ignoreV=None, rootV=1):  
 """  
 执行带数据预处理和标签映射的K-means聚类分析  
   
 参数:  
 data: DataFrame   
 包含待聚类数据的源数据集  
 StrID: str   
 唯一标识列的列名，用于校验数据完整性  
 StrCluster: str   
 需要进行聚类分析的数值列名  
 Knum: int   
 期望的总聚类数（包含忽略值类别）  
 label\_str: list[str], optional   
 自定义聚类标签列表，长度需与有效聚类数匹配  
 ignoreV: float, optional   
 需要单独过滤的特定值（将被归为单独类别）  
 rootV: int, optional   
 指数压缩系数，默认为1表示不压缩  
   
 返回值:  
 DataFrame   
 包含原始ID和聚类结果的两列数据集  
   
 异常:  
 ValueError   
 当存在重复ID或标签数量不匹配时抛出  
 """  
 # 参数校验  
 # 检查ID列唯一性约束  
 **if** data[StrID].duplicated().any():  
 **raise** ValueError("存在重复的ID值")  
 # 验证标签数量与聚类数的逻辑关系  
 **if** label\_str **and** (len(label\_str) != Knum - (1 **if** ignoreV **else** 0)):  
 **raise** ValueError("label\_mapping长度需与有效聚类数一致")  
  
 # 数据预处理  
 # 创建数据副本并进行指数变换  
 df = data[[StrID, StrCluster]].copy()  
 **if** rootV != 1:  
 df[StrCluster] = exponential\_compression(df[StrCluster], rootV)  
  
 # 忽略值处理  
 # 构建有效数据掩码和调整后的聚类数  
 mask = np.ones(len(df), dtype=bool)  
 **if** ignoreV **is** **not** None:  
 mask = (df[StrCluster] != ignoreV)  
 valid\_data = df.loc[mask, StrCluster].values.reshape(-1, 1)  
 n\_clusters = Knum - 1  
 **else**:  
 valid\_data = df[StrCluster].values.reshape(-1, 1)  
 n\_clusters = Knum  
  
 # 核心聚类执行  
 # 使用自动初始化配置的K-means算法  
 kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0, n\_init='auto').fit(valid\_data)  
  
 # 结果列初始化  
 # 创建带掩码的聚类结果列  
 df[StrCluster+'\_judge'] = None  
 df.loc[mask, StrCluster+'\_judge'] = kmeans.labels\_  
  
 # 标签映射处理  
 # 根据聚类中心排序建立标签映射关系  
 cluster\_order = kmeans.cluster\_centers\_.argsort(axis=0).ravel()  
 label\_mapping = {}  
 # 含忽略值的标签分配逻辑  
 **if** label\_str:  
 **if** ignoreV **is** **not** None:  
 **for** i **in** range(len(cluster\_order)):  
 label\_mapping[cluster\_order[i]] = label\_str[i+1]  
 label\_mapping[None] = label\_str[0] # 处理忽略值标签  
 df[StrCluster+'\_judge'] = df[StrCluster+'\_judge'].map(label\_mapping)  
 df.loc[~mask, StrCluster+'\_judge'] = label\_str[0]   
 **else**:  
 **for** i **in** range(len(cluster\_order)):  
 label\_mapping[cluster\_order[i]] = label\_str[i]  
 df[StrCluster + '\_judge'] = df[StrCluster + '\_judge'].map(label\_mapping)  
  
 # 结果分析输出  
 # 准备聚类结果统计信息  
 clustered\_data = df[mask].copy()  
 clustered\_data['cluster'] = kmeans.labels\_  
  
 print("\n\n聚类结果（K-means）：")  
 print("指数压缩值：" + str(rootV))  
  
 # 忽略值信息输出  
 **if** ignoreV **is** **not** None:  
 ignore\_count = (~mask).sum()  
 print(str(label\_str[0]) + f" （{ignore\_count}条）： {ignoreV}")  
 **else**:  
 print("无忽略值")  
  
 # 区间统计输出  
 # 按中心点升序输出各簇统计信息  
 sorted\_clusters = np.argsort(kmeans.cluster\_centers\_.flatten())  
 print("区间（升序排列）：")  
 **for** order, cluster\_idx **in** enumerate(sorted\_clusters, start=1):  
 cluster\_data = clustered\_data[clustered\_data['cluster'] == cluster\_idx][StrCluster]  
 min\_val = cluster\_data.min()  
 max\_val = cluster\_data.max()  
 count = cluster\_data.shape[0]   
  
 # 显示序号调整逻辑  
 display\_order = order **if** **not** ignoreV **else** order-1  
 **if** ignoreV **is** **not** None:  
 display\_order = max(0, display\_order)  
  
 # 标签名称生成逻辑  
 **if** label\_str:  
 label\_name = label\_str[order] **if** ignoreV **else** label\_str[order-1]  
 **else**:  
 label\_name = f"类别 {display\_order}"  
  
 # 显示反压缩后的实际数值区间  
 print(f"{label\_name}（{count}条）区间: [{min\_val\*\*rootV:.2f}, {max\_val\*\*rootV:.2f}]")  
  
 **return** df[[StrID, StrCluster+'\_judge']]

2.3 Jenks自然断裂法

介绍：专门用于一维数据分类的方法，目的是最大化类间差异，最小化类内方差，可以动态规划遍历所有可能分割。

意义：Jenks自然断裂法是一种专为一维数据优化的聚类方法，其核心意义在于通过最小化类内差异并最大化类间差异，精准识别数据中的自然分界点。它特别适用于多峰分布或存在明显密度变化的场景（如地理数据分级），能直观划分出内部同质性强、外部差异显著的类别。相比K-means等通用方法，Jenks对异常值更鲁棒，无需预设分布假设，结果更贴合数据真实结构，为统计分类、地图可视化等领域提供高效且可解释的分级方案，成为一维连续数据分层的黄金标准。

在本次实验中，体验感和K-means差不多，主要起一个多样化分类方法的作用，比较不同分类方法的异同。

**def** **JenkCluster**(data, StrID, StrCluster, Knum, label\_str=None, ignoreV=None, rootV=1):  
 """  
 使用Jenks自然断裂法对数据进行聚类分析  
  
 参数:  
 data (DataFrame): 原始数据集  
 StrID (str): 标识列的名称（需唯一）  
 StrCluster (str): 需要聚类的数值列名称  
 Knum (int): 期望的聚类总数（包含忽略值类别）  
 label\_str (List[str], optional): 聚类标签列表。长度需等于Knum（有忽略值时等于Knum-1）  
 ignoreV (optional): 需要忽略的特殊值，被忽略值将单独归类  
 rootV (float): 指数压缩系数，用于数据预处理（默认1表示不压缩）  
  
 返回:  
 DataFrame: 包含ID列和聚类结果列的二维数据表  
  
 异常:  
 ValueError: 参数校验失败或算法执行错误时抛出  
 """  
 # 参数校验  
 # 检查ID列是否唯一  
 **if** data[StrID].duplicated().any():  
 **raise** ValueError("存在重复的ID值")  
  
 # 数据预处理  
 # 创建数据副本并进行指数压缩  
 df = data[[StrID, StrCluster]].copy()  
 **if** rootV != 1:  
 df[StrCluster] = exponential\_compression(df[StrCluster], rootV)  
  
 # 忽略值处理  
 # 构建有效数据掩码并调整实际聚类数  
 mask = np.ones(len(df), dtype=bool)  
 **if** ignoreV **is** **not** None:  
 mask = (df[StrCluster] != ignoreV)  
 n\_clusters = Knum - 1  
 **else**:  
 n\_clusters = Knum  
  
 # 数据有效性验证  
 # 检查有效数据的多样性  
 valid\_data = df.loc[mask, StrCluster].values.astype(float)  
 **if** len(np.unique(valid\_data)) < 2:  
 **raise** ValueError("Jenks算法需要至少2个不同数值")  
 **if** label\_str **and** (len(label\_str) != Knum - (1 **if** ignoreV **else** 0)):  
 **raise** ValueError("标签数量与聚类数不匹配")  
  
 # Jenks算法核心执行  
 # 获取自然断裂点  
 **try**:  
 breaks = jenkspy.jenks\_breaks(valid\_data, n\_classes=n\_clusters)  
 **except** Exception **as** e:  
 **raise** ValueError(f"Jenks算法执行失败: {str(e)}")  
  
 # 标签分配逻辑  
 # 使用断裂点进行离散化分箱  
 labels = np.searchsorted(breaks, valid\_data, side='right') - 1  
 labels = np.clip(labels, 0, n\_clusters - 1) # 确保极端值落在有效区间内  
  
 # 结果整合  
 # 将聚类结果合并到原始数据  
 df[StrCluster + '\_judge'] = None  
 df.loc[mask, StrCluster + '\_judge'] = labels  
  
 # 标签映射处理  
 # 将数字标签转换为语义化标签  
 **if** label\_str:  
 **if** ignoreV **is** **not** None:  
 label\_mapping = {i: label\_str[i + 1] **for** i **in** range(n\_clusters)}  
 label\_mapping[None] = label\_str[0] # 特殊处理忽略值标签  
 **else**:  
 label\_mapping = {i: label\_str[i] **for** i **in** range(n\_clusters)}  
  
 df[StrCluster + '\_judge'] = df[StrCluster + '\_judge'].map(label\_mapping)  
 **if** ignoreV **is** **not** None:  
 df.loc[~mask, StrCluster + '\_judge'] = label\_str[0]  
  
 # 结果展示  
 # 输出格式化统计信息  
 print("\n\n聚类结果（Jenks自然断裂法）：")  
 print("指数压缩值："+str(rootV))  
  
 **if** ignoreV **is** **not** None:  
 ignore\_count = (~mask).sum()  
 print(str(label\_str[0])+f" （{ignore\_count}条）： {ignoreV}")  
 **else**:  
 print("无忽略值")  
  
 # 区间统计输出  
 # 格式化显示各区间范围和样本数量  
 print("区间（升序排列）：")  
 sorted\_breaks = sorted(breaks)  
 **for** i **in** range(len(sorted\_breaks) - 1):  
 lower = sorted\_breaks[i]  
 upper = sorted\_breaks[i + 1]  
  
 # 区间计数逻辑  
 **if** i == 0:  
 count = ((valid\_data >= lower) & (valid\_data <= upper)).sum()  
 **else**:  
 count = ((valid\_data > lower) & (valid\_data <= upper)).sum()  
  
 # 标签处理逻辑  
 **if** label\_str:  
 **if** ignoreV **is** **not** None:  
 label = label\_str[i + 1] **if** (i + 1) < len(label\_str) **else** f"区间{i + 1}"  
 **else**:  
 label = label\_str[i] **if** i < len(label\_str) **else** f"区间{i + 1}"  
 **else**:  
 label = f"区间{i + 1}"  
  
 # 区间显示格式处理  
 **if** i == len(sorted\_breaks) - 2:  
 interval\_str = f"[{lower\*\*rootV:.2f}, {upper\*\*rootV:.2f}]" # 闭合区间显示  
 **else**:  
 interval\_str = f"({lower\*\*rootV:.2f}, {upper\*\*rootV:.2f}]"  
  
 print(f"{label}（{count}条）: {interval\_str}")  
  
 **return** df[[StrID, StrCluster + '\_judge']]

3分析结果与结论

3.1统计图

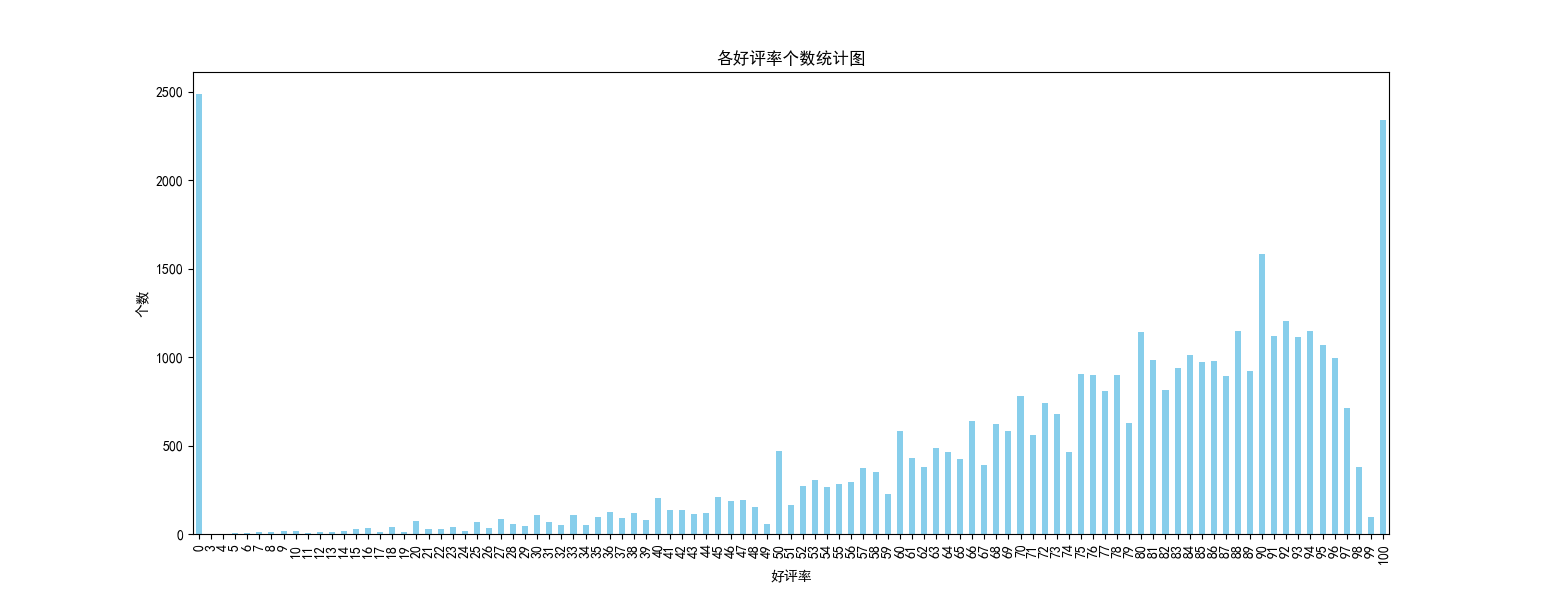


图3.1.1各个好评率对应的个数的柱状图

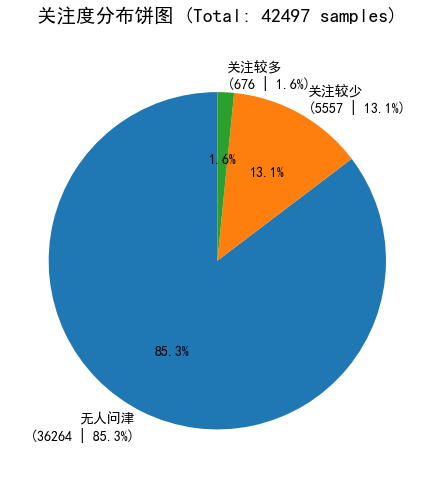


图3.1.2关注度分布饼图（数据来自指数压缩值为3的K-means分类结果）

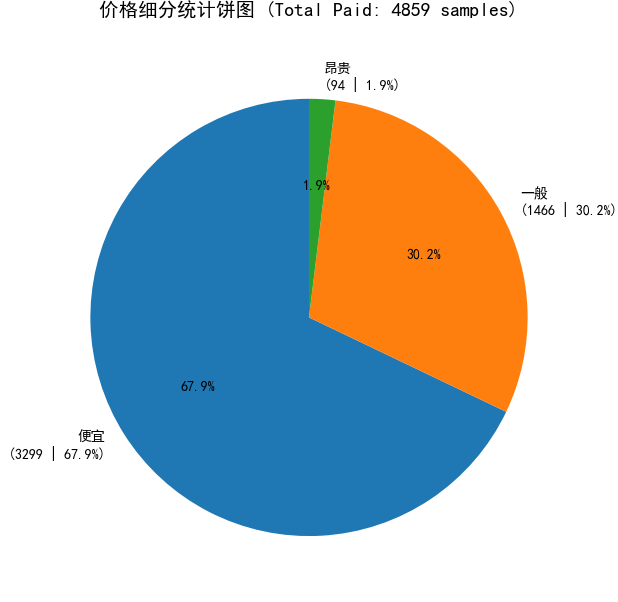
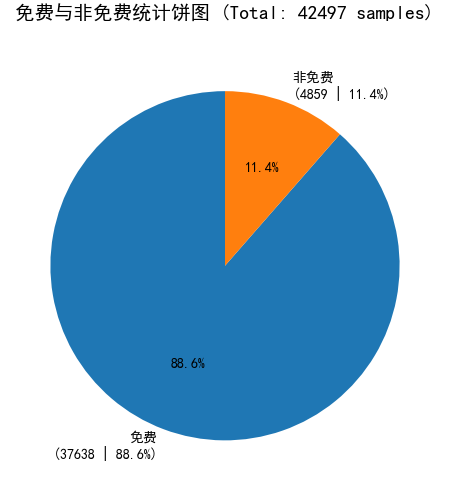


图3.1.3各价格区间分布饼图（数据来自指数压缩值为3的K-means分类结果）

3.2统计表

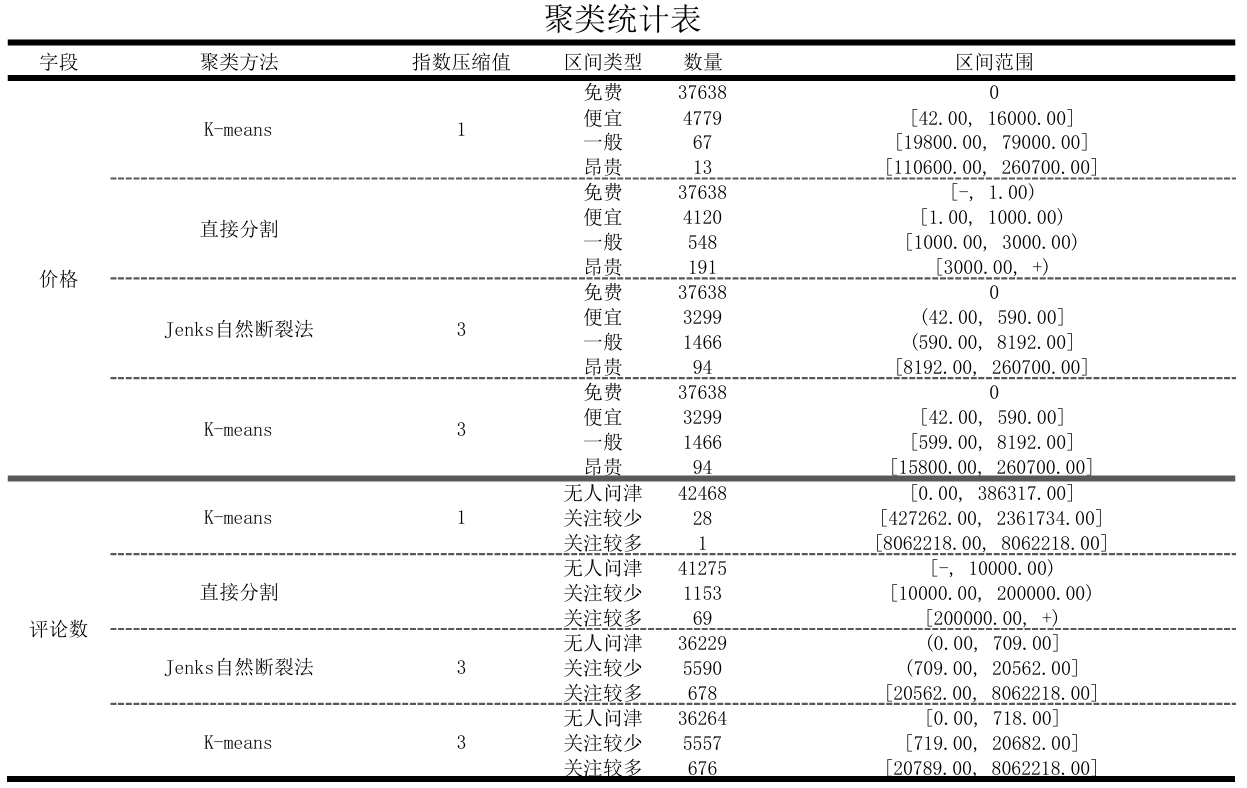


图3.2 聚类统计表

3.3结论

本实验通过数据预处理与三种聚类方法（直接分割法、K-means算法、Jenks自然断裂法）的结合，对游戏价格与评论数据进行了分类分析。实验结果表明，数据预处理与算法选择的协同优化是提升聚类效果的关键。

**（1）数据预处理的重要性**

原始数据存在空值、字符混杂、字段冗余及数值尺度差异过大等问题。通过填充空值、正则清洗、字段筛选及指数压缩（如价格字段取n次根），有效降低了数据噪声与尺度差异对聚类的影响。实验发现，未经压缩的数据在K-means聚类中表现不佳，而指数变换后数据分布更均匀，分类结果显著改善（如图1.2.1与图3.2），说明数据规范化是提升算法鲁棒性的必要步骤。

**（2）聚类方法的对比与适用性**

* **直接分割法**：基于先验知识（如游戏定价常识）设置分割阈值，分类结果直观且符合业务逻辑（如价格区间划分）。但其依赖人工经验，难以捕捉数据内在结构，适用于简单分类或辅助验证其他方法的合理性。
* **K-means算法**：经数据压缩后，其聚类效果显著提升（图3.2）。该方法在多维场景中高效灵活，但对初始簇数敏感且需数据分布接近球形。实验中，压缩后的价格数据通过簇中心排序与标签映射，实现了业务可解释的分类。
* **Jenks自然断裂法**：针对一维数据优化，通过动态规划最小化类内差异，结果更贴合数据自然分界（如评论数的多峰分布）。其优势在于无需预设分布假设，但对高维数据不适用。实验中发现其分类效果与K-means相近，但在解释性上更依赖断裂点分析。

**3. 实验结果与业务价值**

统计图表显示，约90%的游戏属于免费游戏，在非免费游戏中，约70%游戏价格集中于低价区间（＜590卢比），而高价游戏（＞8192卢比）占比不足2%（图3.1.3）。评论数量呈现长尾分布，头部产品占据绝大多数关注度（图3.1.2）。结合聚类结果，可为游戏定价策略、用户画像构建及市场细分提供数据支持。

**4. 局限性与改进方向**

实验局限性包括：依赖单维度聚类，未考虑多字段协同分析，或许可以通过地理探测器来近一步探究字段之间的相关系数；K-means与Jenks对异常值的敏感性未充分探讨；标签映射依赖人工干预。未来可引入轮廓系数评估聚类质量，结合层次聚类或DBSCAN处理复杂分布，并探索自动化标签生成方法以提升效率。