实验一：数据采样方法实践

大数据分析实践实验报告

姓　　名：郑坤武
学　　号：202200150184
班　　级：22 级公信班
学　　院：政治学与公共管理学院

2025 年 9 月 19 日

1 实验目的

- 掌握使用 Pandas 库进行数据处理的基本操作，包括数据读取、空值处理和数据过滤。
- 理解并实践多种数据抽样方法，包括随机抽样、加权抽样和分层抽样。
- 通过实际操作，比较不同抽样方法的特点及其对后续数据分析可能产生的影响。

2 实验环境

- **操作系统：** macOS 15.5
- **编程语言：** Python 3.9
- **开发工具：** Jupyter Notebook

3 具体实验步骤与结果分析

3.1 环境配置与数据读入

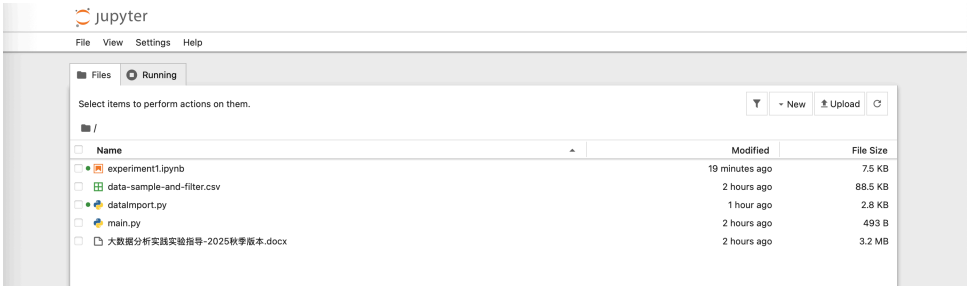
可以在命令行使用如下命令抓取文件，macOS 一般全局采用 UTF-8 进行编码，我检查了这个文件的编码，也修改成了 UTF-8.

```
curl -O http://storage.amesholland.xyz/data.csv
```

Python 的环境就不过多赘述，补充安装一下 Jupyter Notebook，使用 pip 安装 Jupyter Notebook，运行以下命令启动：

```
pip install jupyter
Jupyter Notebook
```

接着会自动跳转打开一个浏览器界面，指向用户家目录，通过设置就可以定位到项目的文件夹中。



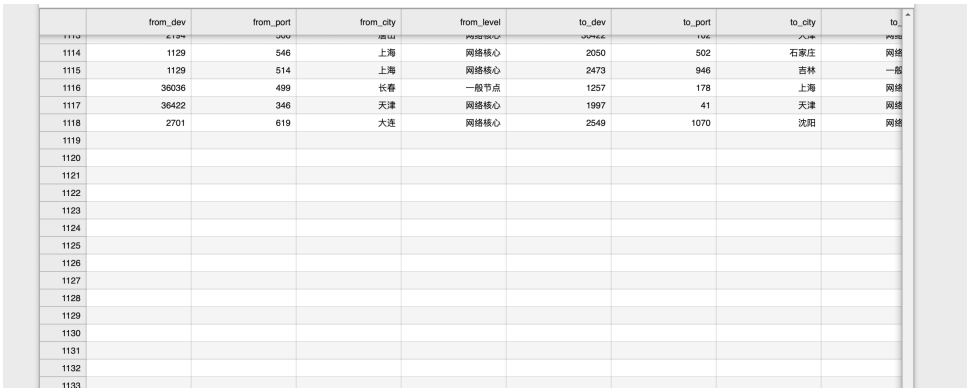
点击右上角的【新建】-> 【Python 3 (ipykernel)】就可以在这个文件上编写代码在线运行，实时查看结果。在 Jupyter Notebook 中可以直接逐段运行代码，只需提前将代码按照逻辑分成多个代码单元格，然后逐个运行。这边新建了一个名为'experiment1' 的文件。

编写代码导入加载先前下载的数据库：

```python
import pandas as pd
from pandas import DataFrame
import numpy as np

# 加载数据
primitive_data = pd.read_csv("data-sample-and-filter.csv", encoding="utf-8")
print(primitive_data)
```

数据成功读入，数据显示在 Jupyter Notebook 中。观察到数据集的底部存在有空行，但在 read_csv 不显示，会被自动跳过。这些无效数据需要在后续步骤中进行清理。



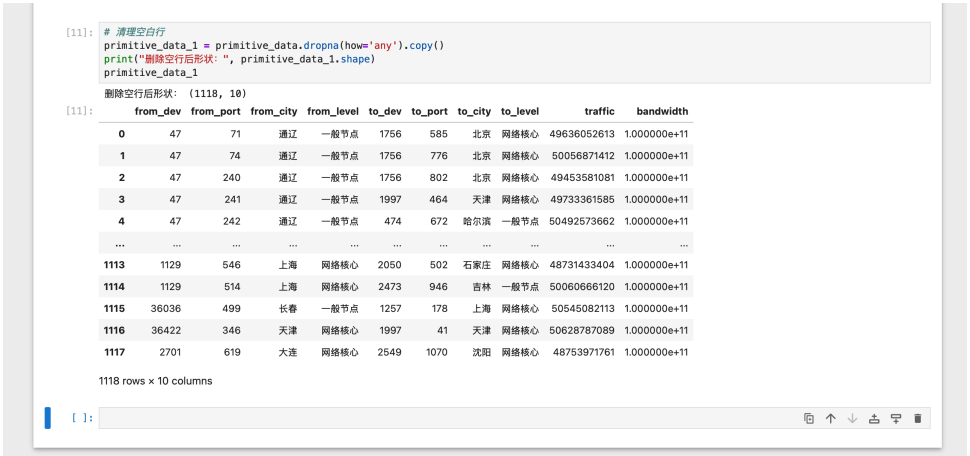
	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_
1114	1129	546	上海	网络核心	2050	502	石家庄	网络
1115	1129	514	上海	网络核心	2473	946	吉林	一般
1116	36036	499	长春	一般节点	1257	178	上海	网络
1117	36422	346	天津	网络核心	1997	41	天津	网络
1118	2701	619	大连	网络核心	2549	1070	沈阳	网络
1119								
1120								
1121								
1122								
1123								
1124								
1125								
1126								
1127								
1128								
1129								
1130								
1131								
1132								
1133								

3.2 数据清洗与过滤

使用 dropna() 方法并指定参数 how='any'，删除所有包含空值的行；执行后，数据集的行数减少，所有空行已被成功删除，数据变得整洁。

```python
# 清理空白行
primitive_data_1 = primitive_data.dropna(how='any').copy()
print("删除空行后形状：", primitive_data_1.shape)
primitive_data_1
```

运行结果如图所示：



```
[11]: # 清理空白行
primitive_data_1 = primitive_data.dropna(how='any').copy()
print("删除空行后形状：", primitive_data_1.shape)
primitive_data_1

删除空行后形状： (1118, 10)
```

[11]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
0	47	71	通辽	一般节点	1756	585	北京	网络核心	49636052613	1.000000e+11
1	47	74	通辽	一般节点	1756	776	北京	网络核心	50056871412	1.000000e+11
2	47	240	通辽	一般节点	1756	802	北京	网络核心	49453581081	1.000000e+11
3	47	241	通辽	一般节点	1997	464	天津	网络核心	49733361585	1.000000e+11
4	47	242	通辽	一般节点	474	672	哈尔滨	一般节点	50492573662	1.000000e+11
...	...	...	...	...	...	...	...	...	...	...
1113	1129	546	上海	网络核心	2050	502	石家庄	网络核心	48731433404	1.000000e+11
1114	1129	514	上海	网络核心	2473	946	吉林	一般节点	50060666120	1.000000e+11
1115	36036	499	长春	一般节点	1257	178	上海	网络核心	50545082113	1.000000e+11
1116	36422	346	天津	网络核心	1997	41	天津	网络核心	50628787089	1.000000e+11
1117	2701	619	大连	网络核心	2549	1070	沈阳	网络核心	48753971761	1.000000e+11

1118 rows × 10 columns

[ ]:

根据实验要求过滤数据，筛选出 traffic 不等于 0 且 from_level 为“一般节点”的记录，作为过滤后的数据。执行以下片段：

```
# 过滤得到 traffic != 0 且 from_level == '一般节点' 但所有数据
data_before_filter = primitive_data_1
mask = (data_before_filter["traffic"] != 0) & (data_before_filter["from_level"] == "一般节点")
data_after_filter_2 = data_before_filter.loc[mask].copy()
print("过滤后形状：", data_after_filter_2.shape)
print(data_after_filter_2["to_level"].value_counts())
data_after_filter_2
```

得到如下结果，总行数变成 550 行：

```
[23]: data_before_filter = primitive_data_1
mask = (data_before_filter["traffic"] != 0) & (data_before_filter["from_level"] == "一般节点")
data_after_filter_2 = data_before_filter.loc[mask].copy()
print("过滤后形状：", data_after_filter_2.shape)
print(data_after_filter_2["to_level"].value_counts())
data_after_filter_2

过滤后形状： (550, 10)
to_level
网络核心    364
一般节点    186
Name: count, dtype: int64
```

[23]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
0	47	71	通辽	一般节点	1756	585	北京	网络核心	49636052613	1.000000e+11
1	47	74	通辽	一般节点	1756	776	北京	网络核心	50056871412	1.000000e+11
2	47	240	通辽	一般节点	1756	802	北京	网络核心	49453581081	1.000000e+11
3	47	241	通辽	一般节点	1997	464	天津	网络核心	49733361585	1.000000e+11
4	47	242	通辽	一般节点	474	672	哈尔滨	一般节点	50492573662	1.000000e+11
...	...	...	...	...	...	...	...	...	...	...
1097	2473	1460	吉林	一般节点	591	586	绥化	一般节点	48409925693	1.000000e+11
1103	36036	18	长春	一般节点	3443	650	青岛	网络核心	48663350759	1.000000e+11
1104	63	6	通辽	一般节点	36036	20	长春	一般节点	50355678076	1.000000e+11
1107	36036	52	长春	一般节点	1129	171	上海	网络核心	49345226162	1.000000e+11
1115	36036	499	长春	一般节点	1257	178	上海	网络核心	50545082113	1.000000e+11

550 rows × 10 columns

经过两步条件过滤，我们得到了一个更符合特定分析目标的数据子集命名为 data_after_filter_2，该数据集将作为后续所有抽样操作的基准数据。

3.3 多种抽样方法实践

以抽取 50 个样本为例，分别实现三种抽样方法。操作前先进行备份，设置一个固定的随机数种子以便于实验复现。

```
# 对数据进行抽样
data_before_sample = data_after_filter_2.copy()
columns = data_before_sample.columns
RANDOM_STATE = 63
N =  len(data_before_sample)
n_sample = 50
```

3.3.1 加权抽样

要求：根据 to_level 的值赋予权重，其中“一般节点”与“网络核心”的权重之比为 1 : 5. 执行：

```
# ===== 1) 加权抽样 (to_level: 一般节点 : 网络核心 = 1 : 5) =====
weight_sample = data_before_sample.copy()
weight_sample["weight"] = np.where(weight_sample["to_level"] == "一般节点", 1, 5)
```

```

weight_sample_finish = weight_sample.sample(
    n=n_sample, weights="weight", random_state=RANDOM_STATE  # 无放回
)[columns].reset_index(drop=True)

print("【加权抽样】结果")
weight_sample_finish
print(weight_sample_finish["to_level"].value_counts())
```

注意到，过滤后原始数据中“网络核心”的数量远多于“一般节点”，那么即使权重是 1:5，由于“网络核心”的基数大，抽到的“网络核心”数量也会很多。加权抽样不是简单地按比例分配样本数量，而是根据权重调整每个个体被抽中的概率。最终我们抽取到了网络核心 45 个，一般节点 5 个：

[19]:
```
# 对数据进行抽样
data_before_sample = data_after_filter_2.copy()
columns = data_before_sample.columns
RANDOM_STATE = 63
N = len(data_before_sample)
n_sample = 50
```

[22]:
```
# ===== 1) 加权抽样 (to_level: 一般节点 : 网络核心 = 1 : 5) =====
weight_sample = data_before_sample.copy()
weight_sample["weight"] = np.where(weight_sample["to_level"] == "一般节点", 1, 5)

weight_sample_finish = weight_sample.sample(
    n=n_sample, weights="weight", random_state=RANDOM_STATE  # 无放回
)[columns].reset_index(drop=True)

print("【加权抽样】结果")
print(weight_sample_finish["to_level"].value_counts())
weight_sample_finish
```

```
【加权抽样】结果
to_level
网络核心    45
一般节点     5
Name: count, dtype: int64
```

[22]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
0	474	671	哈尔滨	一般节点	1756	776	北京	网络核心	51647234796	1.000000e+11
1	63	230	通辽	一般节点	3227	77	济南	网络核心	50504074996	1.000000e+11
2	63	228	通辽	一般节点	235	1506	北京	网络核心	49436165249	1.000000e+11
3	180	276	呼和浩特	一般节点	1997	462	天津	网络核心	51651922009	1.000000e+11
4	180	188	呼和浩特	一般节点	36422	350	天津	网络核心	49047066099	1.000000e+11
5	180	202	呼和浩特	一般节点	1257	536	上海	网络核心	50231972607	1.000000e+11
6	591	15	绥化	一般节点	1385	1490	广州	网络核心	49228307349	1.000000e+11

3.3.2 随机抽样

使用 Pandas 内置的 sample() 方法进行简单随机抽样，随机数设定保持不变。

```
# ===== 2) 随机抽样（简单随机）=====
random_sample_finish = data_before_sample.sample(
    n=n_sample, random_state=RANDOM_STATE
)[columns].reset_index(drop=True)

print("【随机抽样】前5行：")
print(random_sample_finish["to_level"].value_counts())
random_sample_finish
```

随机抽样保证了总体中每一个样本被抽中的概率都是相等的。抽样结果中一般节点和网络核心的比例，大致会接近它们在原始过滤后数据中的总体比例。抽取到了网络核心 34 个，一般节点 16 个。

3.3.3 分层抽样

要求：根据 to_level 这一层进行具体的分层抽样，其中“一般节点”抽 17 个，“网络核心”抽 33 个。

[26]:
```
# ===== 2) 随机抽样（简单随机）=====
random_sample_finish = data_before_sample.sample(
    n=n_sample, random_state=RANDOM_STATE
)[columns].reset_index(drop=True)

print("【随机抽样】前5行：")
print(random_sample_finish["to_level"].value_counts())
random_sample_finish
```

```
【随机抽样】前5行：
to_level
网络核心    34
一般节点    16
Name: count, dtype: int64
```

[26]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
0	180	188	呼和浩特	一般节点	36422	350	天津	网络核心	49047066099	1.000000e+11
1	787	317	玉溪	一般节点	5058	118	南宁	一般节点	49579743371	1.000000e+11
2	96	407	呼和浩特	一般节点	4069	1196	宁波	一般节点	49745162804	1.000000e+11
3	96	123	呼和浩特	一般节点	2841	237	郑州	网络核心	48077485179	1.000000e+11
4	180	276	呼和浩特	一般节点	1997	462	天津	网络核心	51651922009	1.000000e+11
5	180	52	呼和浩特	一般节点	3227	449	济南	网络核心	47569937466	1.000000e+11
6	180	192	呼和浩特	一般节点	591	586	绥化	一般节点	49504348509	1.000000e+11
7	96	134	呼和浩特	一般节点	3643	893	武汉	网络核心	48498103572	1.000000e+11
8	47	417	通辽	一般节点	3227	705	济南	网络核心	49998156282	1.000000e+11
9	96	111	呼和浩特	一般节点	3443	101	青岛	网络核心	51065224623	1.000000e+11
10	47	260	通辽	一般节点	3213	597	重庆	网络核心	50581039842	1.000000e+11
11	63	58	通辽	一般节点	1756	1127	北京	网络核心	51132553467	1.000000e+11
12	96	391	呼和浩特	一般节点	1997	122	天津	网络核心	49100896137	1.000000e+11
13	96	114	呼和浩特	一般节点	2473	769	吉林	一般节点	50350633304	1.000000e+11

```
# ===== 3) 分层抽样（一般节点抽17，网络核心抽33）=====
# 1. 先按层拆分数据
ybjd = data_before_sample.loc[data_before_sample['to_level'] == '一般节点']
wlhx = data_before_sample.loc[data_before_sample['to_level'] == '网络核心']
# 2. 分别从各层中随机抽取指定数量的样本
after_sample = pd.concat([ybjd.sample(17), wlhx.sample(33)])
# 3. 显示结果
print(after_sample["to_level"].value_counts())
after_sample
```

结果如图：

[29]:
```
# ===== 3) 分层抽样（一般节点抽17，网络核心抽33）=====
# 1. 先按层拆分数据
ybjd = data_before_sample.loc[data_before_sample['to_level'] == '一般节点']
wlhx = data_before_sample.loc[data_before_sample['to_level'] == '网络核心']
# 2. 分别从各层中随机抽取指定数量的样本
after_sample = pd.concat([ybjd.sample(17), wlhx.sample(33)])
# 3. 显示结果
print(after_sample["to_level"].value_counts())
after_sample
```

```
to_level
网络核心    33
一般节点    17
Name: count, dtype: int64
```

[29]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
674	591	586	绥化	一般节点	47	243	通辽	一般节点	50565152517	1.000000e+11
59	96	391	呼和浩特	一般节点	47	417	通辽	一般节点	51570663870	1.000000e+11
72	180	42	呼和浩特	一般节点	36539	1140	杭州	一般节点	49293665157	1.000000e+11
129	474	1410	哈尔滨	一般节点	4069	1205	宁波	一般节点	46523775334	1.000000e+11
1039	180	264	呼和浩特	一般节点	36036	54	长春	一般节点	49124032697	1.000000e+11
180	787	360	玉溪	一般节点	3615	191	长沙	一般节点	49629725686	1.000000e+11
86	180	226	呼和浩特	一般节点	36036	20	长春	一般节点	49248544673	1.000000e+11
390	474	683	哈尔滨	一般节点	2473	762	吉林	一般节点	50437152432	1.000000e+11
775	96	134	呼和浩特	一般节点	180	98	呼和浩特	一般节点	51993612239	1.000000e+11
127	474	1399	哈尔滨	一般节点	4360	468	南京	一般节点	50372436809	1.000000e+11
604	96	134	呼和浩特	一般节点	2473	1460	吉林	一般节点	49201392181	1.000000e+11
962	4448	127	无锡	一般节点	47	425	通辽	一般节点	50961073987	1.000000e+11
151	591	586	绥化	一般节点	180	192	呼和浩特	一般节点	49061517661	1.000000e+11
779	96	152	呼和浩特	一般节点	180	202	呼和浩特	一般节点	51162997127	1.000000e+11
423	591	558	绥化	一般节点	180	20	呼和浩特	一般节点	48364223310	1.000000e+11

3.3.4 其他抽样方法思路

比如还可以尝试一下等间距抽样，先计算需要的间距，然后选择一个起始点进行等间距抽样。运行如下片段：

```python
# ===== 4) 系统抽样（等距抽样）=====
# 计算抽样间隔
k = N // n_sample
# 随机选择起始点
start = np.random.randint(0, k)
# 生成抽样索引
systematic_indices = [start + i*k for i in  range(n_sample) if start + i*k < N]
# 根据索引抽取样本
systematic_sample = data_before_sample.iloc[systematic_indices].copy()
systematic_sample = systematic_sample[columns].reset_index(drop=True)

print("【系统抽样】结果")
print(f"抽样间隔k: {k}")
print(f"起始点: {start}")
print(systematic_sample["to_level"].value_counts())
systematic_sample
```

结果如图：

[30]:
```
【系统抽样】结果
抽样间隔k: 11
起始点: 8
to_level
网络核心    27
一般节点    23
Name: count, dtype: int64
```

[30]:

	from_dev	from_port	from_city	from_level	to_dev	to_port	to_city	to_level	traffic	bandwidth
0	47	251	通辽	一般节点	2549	839	沈阳	网络核心	50755299504	1.000000e+11
1	63	12	通辽	一般节点	180	252	呼和浩特	一般节点	49290094443	1.000000e+11
2	63	232	通辽	一般节点	36422	124	天津	网络核心	49752623185	1.000000e+11
3	96	120	呼和浩特	一般节点	1997	250	天津	网络核心	50700267269	1.000000e+11
4	96	157	呼和浩特	一般节点	2050	443	石家庄	网络核心	50096366926	1.000000e+11
5	180	10	呼和浩特	一般节点	2994	270	洛阳	网络核心	50401714739	1.000000e+11
6	180	52	呼和浩特	一般节点	63	286	通辽	一般节点	49155371449	1.000000e+11
7	180	218	呼和浩特	一般节点	3443	650	青岛	网络核心	50106572586	1.000000e+11
8	474	360	哈尔滨	一般节点	2473	946	吉林	一般节点	51819320173	1.000000e+11

4 实验总结与收获

通过本次实验，我熟练掌握了数据清洗的关键技能，使用 dropna() 方法有效清除了数据集中的空值，并通过 loc[] 结合条件语句就能精准筛选出符合要求的数据。在抽样方法实践中，我深刻体会到不同方法的特点：随机抽样的简单高效让我快速获取代表性样本；加权抽样让我学会通过权重设置灵活调整抽样概率，突出重要群体；分层抽样则让我理解了如何通过分层保证关键特征的样本代表性。没有通用的”最佳”抽样方法，只有针对具体场景的”最合适”选择。不同的研究目的需

要配以不同的抽样策略。

附录 Appendix

完整代码 dataImport.py :

```python
import pandas as pd
from pandas import DataFrame
import numpy as np

# 加载数据
primitive_data = pd.read_csv("data-sample-and-filter.csv", encoding="utf-8")
# print(primitive_data.head())

# 清理空白行
primitive_data_1 = primitive_data.dropna(how='any').copy()
print("删除空行后形状：", primitive_data_1.shape)
primitive_data_1

# 过滤得到 traffic != 0 且 from_level == '一般节点' 但所有数据
data_before_filter = primitive_data_1
mask = (data_before_filter["traffic"] != 0) & (data_before_filter["from_level"] == "一般节点")
data_after_filter_2 = data_before_filter.loc[mask].copy()
print("过滤后形状：", data_after_filter_2.shape)
data_after_filter_2

# 对数据进行抽样
data_before_sample = data_after_filter_2.copy()
columns = data_before_sample.columns
RANDOM_STATE = 63
N =  len(data_before_sample)
n_sample = 50


# ===== 1) 加权抽样（to_level: 一般节点 : 网络核心 = 1 : 5）=====
weight_sample = data_before_sample.copy()
weight_sample["weight"] = np.where(weight_sample["to_level"] == "一般节点", 1, 5)

weight_sample_finish = weight_sample.sample(
    n=n_sample, weights="weight", random_state=RANDOM_STATE  # 无放回
)[columns].reset_index(drop=True)

print("【加权抽样】结果")
print(weight_sample_finish["to_level"].value_counts())
weight_sample_finish

# ===== 2) 随机抽样（简单随机）=====
random_sample_finish = data_before_sample.sample(
    n=n_sample, random_state=RANDOM_STATE
)[columns].reset_index(drop=True)
```

```
45
46 print("【随机抽样】前5行：")
47 print(random_sample_finish["to_level"].value_counts())
48 random_sample_finish
49
50 # ===== 3) 分层抽样（一般节点抽17，网络核心抽33）=====
51 # 1. 先按层拆分数据
52 ybjd = data_before_sample.loc[data_before_sample['to_level'] == '一般节点']
53 wlhx = data_before_sample.loc[data_before_sample['to_level'] == '网络核心']
54 # 2. 分别从各层中随机抽取指定数量的样本
55 after_sample = pd.concat([ybjd.sample(17), wlhx.sample(33)])
56 # 3. 显示结果
57 print(after_sample["to_level"].value_counts())
58 after_sample
59
60 # ===== 4) 系统抽样（等距抽样）=====
61 # 计算抽样间隔
62 k = N // n_sample
63 # 随机选择起始点
64 start = np.random.randint(0, k)
65 # 生成抽样索引
66 systematic_indices = [start + i*k for i in  range(n_sample) if start + i*k < N]
67 # 根据索引抽取样本
68 systematic_sample = data_before_sample.iloc[systematic_indices].copy()
69 systematic_sample = systematic_sample[columns].reset_index(drop=True)
70
71 print("【系统抽样】结果")
72 print(f"抽样间隔k：{k}")
73 print(f"起始点：{start}")
74 print(systematic_sample["to_level"].value_counts())
75 systematic_sample
```