第 5 章

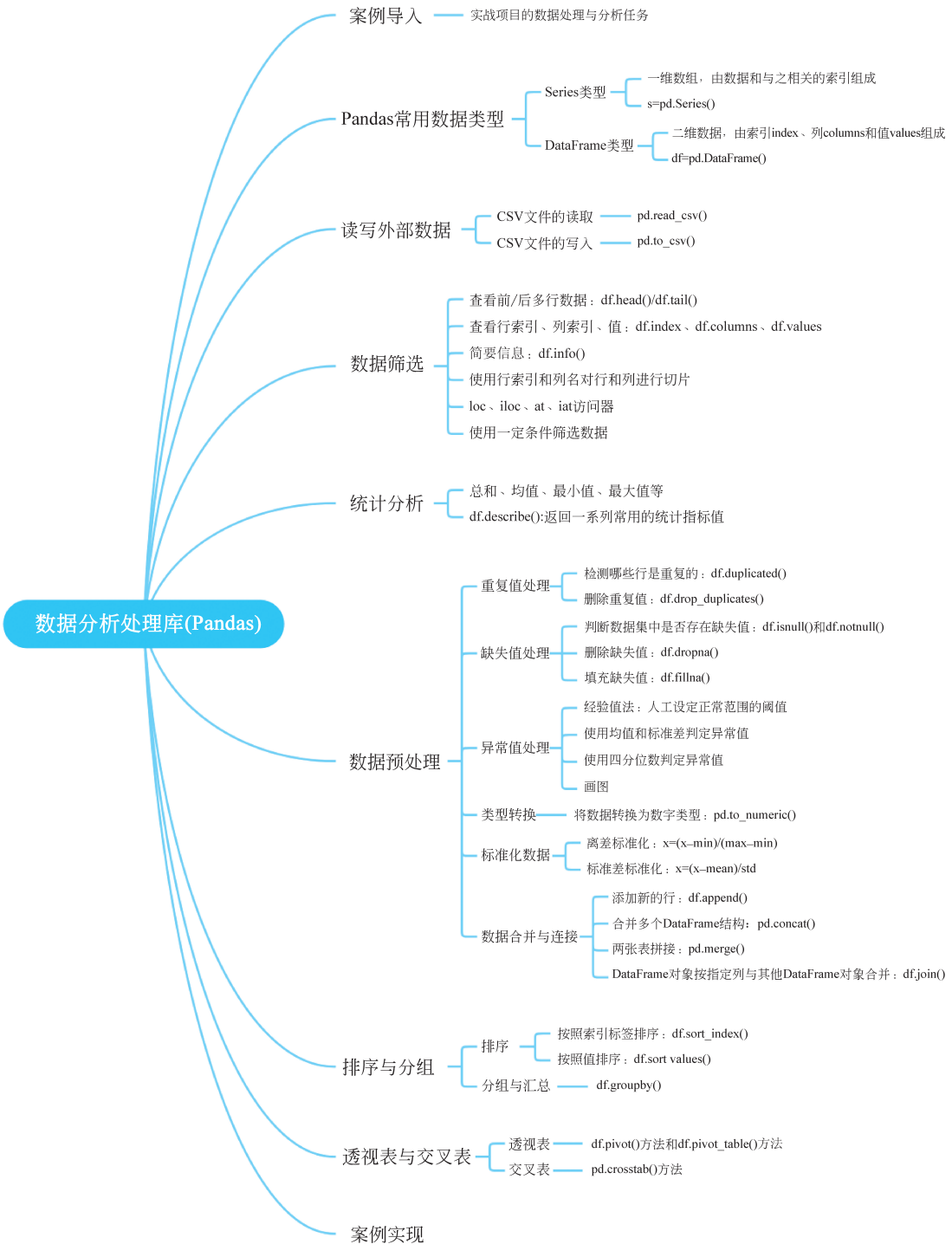
数据分析处理库(Pandas)

数据分析是指用适当的统计、分析方法对收集来的大量数据进行分析,将它们加以汇总和理解,以求最大程度发挥数据的作用。Python 是近年来最常用的数据分析语言之一,而 Pandas 是用于数据分析的最好的第三方库,支持处理从不同的数据源收集而来的索引数据。Pandas 基于 Numpy 和 Matplotlib 库,提供了便于操作数据的数据类型和大量的数据分析函数,这使得 Pandas 成为高效且强大的数据分析库。本章介绍使用 Pandas 完成数据生成和导入、数据预处理、数据筛选、排序与分组、透视等常见操作。

本章学习目标

- 理解并学会应用 Pandas 的常用数据类型。
- 掌握读写外部数据的方法。
- 掌握数据筛选和数据统计的方法。
- 掌握数据预处理的方法,包括重复值、缺失值和异常值的处理,数据类型的转换,数据的标准化操作,数据的合并和连接方法。
- 掌握数据的排序和分组。
- 理解透视表与交叉表。
- 掌握使用 Pandas 实现实战项目中的数据分析任务。

本章思维导图


数据分析处理库(Pandas)
案例导入
实战项目的数据处理与分析任务
Pandas常用数据类型
Series类型
一维数组，由数据和与之相关的索引组成
s=pd.Series()
DataFrame类型
二维数据，由索引index、列columns和值values组成
df=pd.DataFrame()
读写外部数据
CSV文件的读取
pd.read_csv()
CSV文件的写入
pd.to_csv()
数据筛选
查看前/后多行数据：df.head()/df.tail()
查看行索引、列索引、值：df.index、df.columns、df.values
简要信息：df.info()
使用行索引和列名对行和列进行切片
loc、iloc、at、iat访问器
使用一定条件筛选数据
统计分析
总和、均值、最小值、最大值等
df.describe():返回一系列常用的统计指标值
数据预处理
重复值处理
检测哪些行是重复的：df.duplicated()
删除重复值：df.drop_duplicates()
缺失值处理
判断数据集中是否存在缺失值：df.isnull()和df.notnull()
删除缺失值：df.dropna()
填充缺失值：df.fillna()
异常值处理
经验值法：人工设定正常范围的阈值
使用均值和标准差判定异常值
使用四分位数判定异常值
画图
类型转换
将数据转换为数字类型：pd.to_numeric()
标准化数据
离差标准化：x=(x–min)/(max–min)
标准差标准化：x=(x–mean)/std
数据合并与连接
添加新的行：df.append()
合并多个DataFrame结构：pd.concat()
两张表拼接：pd.merge()
DataFrame对象按指定列与其他DataFrame对象合并：df.join()
排序与分组
排序
按照索引标签排序：df.sort_index()
按照值排序：df.sort values()
分组与汇总
df.groupby()
透视表与交叉表
透视表
df.pivot()方法和df.pivot_table()方法
交叉表
pd.crosstab()方法
案例实现

5.1　案例导入


扫一扫

在综合实战项目中,“北京链家网”租房数据的抓取任务已在第 3 章完成,得到了数据表 bj_lianJia.csv,如图 5-1 所示。该数据表包含 ID、城区名(district)、街道名(street)、小区名(community)、楼层信息(floor)、有无电梯(lift)、面积(area)、房屋朝向(toward)、户型(model)、房租(rent)等信息。本章以此数据表的数据处理与分析为主线,讲解 Pandas 库的常用操作,主要包括以下内容。

(1) 重复行的处理。

(2) 缺失值的处理。

(3) 内容格式清洗。

(4) 属性重构造。

(5) 对房租 rent 列数据进行统计分析。

ID	floor	lift	district	street	community	area	toward	model	rent
rent0001	中楼层/6层	无	房山	良乡	行宫园二里	85.00㎡	南 北	2室2厅1卫	3500
rent0002	低楼层/17层	有	顺义	顺义其它	尚峯壹號	107.00㎡	南	3房间1卫	5400
rent0003	中楼层/6层	无	大兴	西红门	同兴园	72.00㎡	南 北	2室1厅1卫	3800
rent0004	中楼层/8层	有	顺义	后沙峪	智地香蜜湾	71.13㎡	东	3房间2卫	6900
rent0005	中楼层/4层	有	朝阳	酒仙桥	丽都壹号	54.41㎡	东	2室1厅1卫	9800
rent0006	高楼层/12层	有	朝阳	十八里店	江南山水	132.00㎡	南 北	3室2厅2卫	10000
rent0007	高楼层/6层	无	大兴	观音寺	宇丰苑	123.80㎡	南 北	3室1厅1卫	4500
rent0008	高楼层/16层	有	昌平	天通苑	天通苑中苑	152.81㎡	南 北	3室1厅2卫	10000
rent0009	低楼层/6层	无	昌平	东关	东关南里小区	81.22㎡	南 北	3房间1卫	
rent0010	中楼层/12层	有	大兴	大兴新机场	龙熙公馆	113.28㎡	南	3房间1卫	4000

图 5-1　“北京链家网”租房数据表的部分数据展示

5.2　Pandas 常用数据类型

Pandas 库提供了一种扩展的数据类型,关注数据与索引之间的关系,索引项是 Pandas 数据类型最独特的地方。Pandas 常用的数据类型包括表示一维数组结构的 Series 类型和二维数据结构的 DataFrame 类型。为了方便展示结果,本节主要使用 IDLE 运行代码。

5.2.1　Series 类型

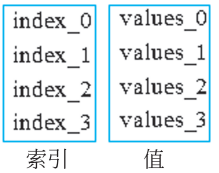


图 5-2　Series 类型的组成部分


扫一扫

Series 类型是 Pandas 提供的一维数组结构,由数据和与之相关的索引组成,其结构如图 5-2 所示。

(1) Series 数组的创建。使用列表、标量值、字典、ndarray 对象等数据创建 Series 一维数组。

① 自动生成索引。在创建 Series 数组时,如果没有给定索引,则可以自动生成从 0 开始的非负整数作为索引。示例如下。

```
>>> import pandas as pd
>>> s1=pd.Series(['海淀', '房山', '顺义', '大兴'])
```

```
>>> s1
0    海淀
1    房山
2    顺义
3    大兴
dtype: object
```

② 自定义索引。在创建 Series 数组时,可以使用 index 参数自行设定索引,示例如下。

```
>>> s2=pd.Series(['海淀', '房山', '顺义', '大兴'], index=['a', 'b', 'c', 'd'])
>>> s2
a    海淀
b    房山
c    顺义
d    大兴
dtype: object
```

③ 使用标量值创建 Series 数组。使用一个标量值创建 Series 数组,需要注意的是,如果在创建数组时没有给出 index 参数,则自动生成 0 这一索引项,如下面代码的 s3 变量的索引为 0。如果在创建数组时给出 index 参数,如 s4 变量,其索引项为 index 参数的值,数据项为重复的标量值。可以看出,index 参数决定了 Series 数组大小。

```
>>> s3=pd.Series('海淀')
>>> s3
0    海淀
dtype: object
>>> s4=pd.Series('海淀',index=['a','b','c', 'd'])
>>> s4
a    海淀
b    海淀
c    海淀
d    海淀
dtype: object
```

④ 使用字典创建 Series 数组。字典本身就是一种“键:值”对组成的数据类型,所以由字典创建 Series 类型时,字典的键作为 Series 的索引项,字典的值作为数据项,例如下面代码的 s5 变量。如果使用字典创建 Series 类型时给出 index 参数,得到的 Series 数组会根据 index 参数值作为键,从字典中选择相应的值组成,如果字典中没有相应的键,那么得到的 Series 数组中相应索引对应的值为 NaN,例如变量 s6 中的“d”索引项对应的值为 NaN。NaN 的意思是 not a number。可以看出,使用字典创建 Series 类型时,index 参数决定了 Series 数组的大小和顺序。

```
>>> s5=pd.Series({'a':'海淀', 'b':'房山', 'c':'顺义', 'd':'大兴'})
>>> s5
a    海淀
b    房山
c    顺义
d    大兴
```

```
dtype: object
>>> s6=pd.Series({'a': '海淀','b': '房山','c': '顺义'}, index=['c','a','b',
'd'])
>>> s6
c    顺义
a    海淀
b    房山
d    NaN
dtype: object
```

⑤ 使用 ndarray 对象创建 Series 数组。导入 Numpy 库后，Series 类型的索引和数据都可以通过 ndarray 类型创建。示例如下。

```
>>> import numpy as np
>>> s7=pd.Series(np.random.randint(0,20,5), index=np.arange(5))
>>> s7
0     9
1     4
2    19
3     4
4     2
dtype: int32
```

(2) Series 类型的常用操作。包括 Series 数据访问、数据的运算和对齐操作。

① Series 类型由索引 index 和值 values 两部分组成，可以使用“对象名.index”获取 Series 对象的索引部分，使用“对象名.values”获取 Series 对象的数据部分。代码如下。


扫一扫

```
>>> s2=pd.Series(['海淀', '房山', '顺义', '大兴'], index=['a', 'b', 'c', 'd'])
>>> s2.index
Index(['a', 'b', 'c', 'd'], dtype='object')
>>> s2.values
array(['海淀', '房山', '顺义', '大兴'], dtype=object)
```

② 使用索引访问 Series 对象。可以使用自定义索引或自动索引访问数组的值或对数组进行切片，也可以像字典一样使用 get()方法访问数据。代码如下。

```
>>> s2=pd.Series(['海淀', '房山', '顺义', '大兴'], index=['a', 'b', 'c', 'd'])
>>> s2['a']          #使用自定义索引访问数组的值
'海淀'
>>> s2[['a', 'c']]   #使用自定义索引对数组进行切片
a    海淀
c    顺义
dtype: object
>>> s2[1]            #使用自动索引访问数组的值
'房山'
>>> s2[1:3]          #使用自动索引对数组进行切片
b    房山
c    顺义
```

```
dtype: object
>>> s2.get('a')        #使用 get 函数访问数组的值,类似字典操作
'海淀'
```

③ 使用 Python 内置函数或 Series 本身提供的方法操作 Series 对象。代码如下。

```
>>> import numpy as np
>>> s8=pd.Series(np.random.randint(0,100,5), index=['a', 'b', 'c', 'd', 'e'])
>>> s8
a    16
b     0
c     8
d    96
e    49
dtype: int32
>>> max(s8)            #使用 Python 内置函数
96
>>> s8/2               #使用运算符,运算对象是 Series 的 values,运算结果是 Series 类型
a     8.0
b     0.0
c     4.0
d    48.0
e    24.5
dtype: float64
>>> np.floor(s8/2)    #使用 Numpy 提供的向下取整函数,运算对象是 Series 的 values
a     8.0
b     0.0
c     4.0
d    48.0
e    24.0
dtype: float64
>>> s8.median()       #使用 Series 对象提供的求中值函数
16.0
>>> s8[s8>50]         #数据筛选
d    96
dtype: int32
```

④ Series 类型的对齐操作。两个 Series 对象进行运算时,Pandas 会自动对齐不同索引的数据。这一操作和 Numpy 数组操作不同。如果 x 和 y 是不同形状的 ndarray 类型,只有符合广播条件时才能计算,否则会报错。但如果 x 和 y 是不同形状的 Series 类型,则以索引值自动对齐:相同索引的值进行运算,如果一个数组中没有某个索引值,则运算结果为 NaN。示例如下。

```
>>> x=np.array([1,2,3,4])       #x 变量是 ndarray 类型
>>> y=np.array([5,6,7])         #y 变量是 ndarray 类型
>>> x + y
Traceback(most recent call last):
  File "<pyshell #26>", line 1, in <module>
    x + y
```

```
ValueError: operands could not be broadcast together with shapes (4,) (3,)
>>> x=pd.Series([1,2,3,4])      #x变量是Series类型
>>> y=pd.Series([5,6,7])        #y变量是Series类型
>>> x + y
0     6.0
1     8.0
2    10.0
3     NaN
dtype: float64
```

5.2.2　DataFrame 类型

DataFrame 类型表示二维数据，可以将其看作一个二维表格，其结构由 3 部分组成：索引 index、列 columns 和值 values，如图 5-3 所示。

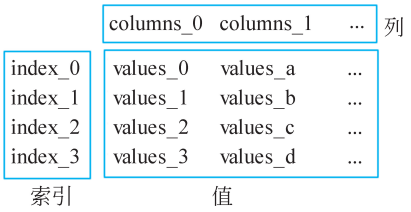


图 5-3　DataFrame 结构的组成部分

Pandas 支持两种形式创建 DataFrame 类型数据：一是使用代码直接创建 DataFrame 数据，二是从外部数据读入到 DataFrame 类型。本节重点讲解使用代码直接创建 DataFrame 数据。

(1) 使用字典创建。字典的键作为 DataFrame 对象的列名，字典的值作为 DataFrame 对象的值，如果指定 DataFrame 对象的索引部分，则需要用到 index 参数。示例如下。

```
>>> df1=pd.DataFrame({'district':['海淀','房山','顺义','大兴'], 'street':['清河
','良乡','后沙峪','西红门'],'rent':[8500,3500,6900,3800]}, index=['a', 'b', 'c',
'd'])
>>> df1
       district  street  rent
a       海淀        清河      8500
b       房山        良乡      3500
c       顺义        后沙峪    6900
d       大兴        西红门    3800
```

(2) 使用 ndarray 对象创建。如果不给定 index 和 columns 参数，将自动生成从 0 开始的非负整数，如下面代码中的 df2。也可以自定义 index 和 columns，如 df3。

```
>>> df2=pd.DataFrame(np.arange(12).reshape(3,4))
>>> df2
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
>>> df3=pd.DataFrame(np.arange(12).reshape(3,4), index=np.arange(3),columns
=['a','b','c','d'])
>>> df3
   a  b   c   d
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

5.3 读写外部数据

在现实中,数据通常存储在外部文件中,Pandas 提供了多种方法读取和写入数据,具体可以查看 Pandas 官网(https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html)。Pandas 常用的几种文件读写方法如表 5-1 所示。本节重点讲解 CSV 文件的读写方法。

表 5-1 Pandas 中常用的几种数据文件读写方法

数据类型	数据文件	读方法	写方法
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	XML	read_xml	to_xml
text	HTML	read_html	to_html
binary	MS Excel	read_excel	to_excel
SQL	SQL	read_sql	to_sql


扫一扫

5.3.1 CSV 文件的读取

Pandas 使用 read_csv()方法读取 csv 文件中的数据。read_csv()方法有多个参数,大多数参数使用默认值即可,这里介绍几个比较常用的参数。

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, header='infer',
names=NoDefault.no_default, index_col=None, usecols=None, converters=None,
skiprows=None, encoding=None)
```

参数说明如下。

① filepath_or_buffer:文件所在路径,可以是字符串形式的文件路径、url 或文件对象。这个参数是唯一一个必传的参数。

② sep:指定数据集中各字段之间的分隔符,默认为一个英文逗号,即“,”。如果分隔符指定错误,在读取数据的时候,每一行数据将连成一片。

③ header:指定由哪一行作为列名,默认为 header=0,表示第一行作为列名。如果 header=None,表示不从文件数据中指定行作为列名,这时 Pandas 会自动生成从零开始的序列作为列名。

④ names:表示为读取的列加上指定的列名。

⑤ index_col:设置原 csv 文件里哪一列的值作为 DataFrame 对象的 index 索引值。index 默认从 0 开始的自动索引。

⑥ usecols:指定需要读取原数据集中的哪些列,可以是列名,也可以是列序号。

⑦ converters:接收字典形式,用于转换 csv 表中某些列中的值。键可以是列名,也可以是列序号。

⑧ skiprows：数据读取时，指定需要跳过原数据集开头的行数。

⑨ encoding：表示文件的编码格式，常用的编码有 UTF-8、UTF-16、GBK、GB2312、GB18030 等，通常为“UTF-8”。如果文件中含有中文，有时需要指定字符编码。

下面演示指定 read_csv()方法中的不同参数从“北京链家网”租房数据表 bj_lianJia.csv 中读取数据，并输出结果。

(1) 指定 encoding 和 usecols 参数，示例如下。

```
import pandas as pd
df1=pd.read_csv('bj_lianJia.csv', encoding='gbk', usecols=['ID','floor',
'lift','district','area','rent'])
print(df1)
```

输出结果为：

```
             ID    floor          lift    district    area      rent
0       rent0001   中楼层/6层      无       房山        85.00m²   3500.0
1       rent0002   低楼层/17层     有       顺义        107.00m²  5400.0
2       rent0003   中楼层/6层      无       大兴        72.00m²   3800.0
        ...        ...            ...      ...         ...       ...
4338    rent4339   高楼层/28层     有       朝阳        100.00m²  9000.0
4339    rent4340   低楼层/2层      无       怀柔        194.21m²  18000.0
4340    rent4341   低楼层/4层      无       通州        188.00m²  13000.0
[4341 rows x 6 columns]
```

(2) 指定 encoding、usecols 和 index_col 参数。其中，usecols 使用列序号，index_col 设置为“ID”。此例中的 df2 与上例中的 df1 相比，df2 的 index 为原表的“ID”列，而 df1 没有指定 index_col，默认为从 0 开始的自动索引。示例如下。

```
df2=pd.read_csv('bj_lianJia.csv', encoding='gbk', usecols=[0,1,2,3,6,9],index
_col='ID')
print(df2[:5])
```

输出结果为：

```
ID           floor          lift    district    area      rent
rent0001     中楼层/6层      无       房山        85.00m²   3500.0
rent0002     低楼层/17层     有       顺义        107.00m²  5400.0
rent0003     中楼层/6层      无       大兴        72.00m²   3800.0
rent0004     中楼层/8层      有       顺义        71.13m²   6900.0
rent0005     中楼层/4层      有       朝阳        54.41m²   9800.0
```

(3) 指定 encoding、usecols、names 和 skiprows 参数。其中，names 参数为 df3 自定义的列名，skiprows 参数为 1，表示略过原数据表的第 1 行，即原列名所在的行。示例如下。

```
df3=pd.read_csv('bj_lianJia.csv', encoding='gbk', usecols=[0,1,2,3,6,9],names
=['编号','楼层','有无电梯','城区','面积','房租'],skiprows=1)
print(df3[:5])
```

输出结果为：

```
         编号       楼层        有无电梯   城区   面积       房租
0   rent0001   中楼层/6层      无      房山   85.00m²    3500.0
1   rent0002   低楼层/17层     有      顺义   107.00m²   5400.0
2   rent0003   中楼层/6层      无      大兴   72.00m²    3800.0
3   rent0004   中楼层/8层      有      顺义   71.13m²    6900.0
4   rent0005   中楼层/4层      有      朝阳   54.41m²    9800.0
```

扫一扫

5.3.2 CSV 文件的写入

to_csv()方法可以将 Pandas 数据写入到文本文件或 csv 文件中。下面给出 to_csv()方法中几个比较常用的参数，参数的含义与 read_csv()方法相似。

```
DataFrame.to_csv(path_or_buf=None, sep=',', columns=None, header=True, index=
True, encoding=None)
```

下面演示 to_csv()方法中不同参数的使用。

（1）保存为 txt 文件，分隔符为“\t”。代码如下，保存的文件内容如图 5-4 所示。

```
df4=pd.DataFrame({'district':['海淀','房山','顺义','大兴'], 'street':['清河','良
乡','后沙峪','西红门'],'rent':[8500,3500,6900,3800]}, index=['a', 'b', 'c', 'd'])
print(df4)
df4.to_csv('save_df4.txt',sep='\t')
```

输出结果为：

```
       district          street       rent
a      海淀     清河       8500
b      房山     良乡       3500
c      顺义     后沙峪     6900
d      大兴     西红门     3800
```

图 5-4 保存为 txt 文件，分隔符为“\t”

（2）保存为 txt 文件，分隔符使用默认分隔符“,”，指定 index 参数为 False，表示不保存 DataFrame 中的 index 项。代码如下。保存的文件内容如图 5-5 所示。

```
df4.to_csv('save_df4.txt',index=False)
```

（3）保存为 csv 文件，指定 header 参数，表示保存文件时重新设置列名。代码如下，保