数据预处理

第4章　数据预处理

目录
CONTENTE

4.1 数据清洗

4.2 合并连接与重塑

4.3 数据变换

引言

数据预处理是一项极其重要又十分繁琐的工作，数据预处理的好坏对数据分析结果有决定性作用。

在实际的数据分析和建模中，大约80%的时间是花费在数据准备和预处理上。

第4章 数据预处理

Python

4.1 数据清洗

第五章 数据预处理

- 数据清洗主要是处理原始数据中的**重复数据**、**缺失数据**和**异常数据**，使数据分析不受无效数据的影响。
  - 重复数据一般可删除；
  - 缺失数据可删除或填充；
  - 异常值可以删除或修正。

4.1.1 重复数据的处理

（1）检测重复值

DataFrame.duplicated(subset = None，keep ='first')

- **参数：**
  - subset：**默认情况下根据每一行的所有值来判断重复**，若需指定其中几列来判断重复，则设置此参数。
  - keep：{'first'，'last'，False}，默认为'first'
    - first：从前往后标记重复项，第一次出现的数据标记为False，其余与之重复的都标记为True。
    - last：与first相反。
    - False：将所有重复项都标记为True。

[示例] 从前往后（默认）检测和标记完全重复的行

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('--------------------')
# 第一次出现的行标记为False，其后重复行标记为True
d = df.duplicated()
print(d)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
0    False
1    False
2     True
3    False
4    False
5     True
dtype: bool
```

[示例] 从后往前检测和标记完全重复的行

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('-'*24)
# 第一次出现的行标记为False，其后重复行标记为True
d = df.duplicated(keep='last')
print(d)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
------------------------
0    True
1    False
2    False
3    False
4    True
5    False
dtype: bool
```

[示例] 只要是重复的行就标记为True

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('------------------')
d = df.duplicated(keep=False)
print(d)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
------------------
0     True
1    False
2     True
3    False
4     True
5     True
dtype: bool
```

[示例] 仅选择某一列来检测重复值

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('----------------------')
# 根据brand列的值检测重复行
d = df.duplicated(subset='brand')
print(d)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
0    False
1    False
2     True
3     True
4    False
5     True
dtype: bool
```

[示例] 仅选择某几列来检测重复值

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('----------------------')
# 根据brand列和rating列的值检测重复行
d = df.duplicated(subset=['brand','style'])
print(d)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
0    False
1    False
2     True
3     True
4    False
5     True
dtype: bool
```

[应用实例] 使用duplicated()函数识别可能由支付系统重试导致的重复交易。

```
import pandas as pd

df = pd.read_csv('duplicated.csv')
duplicates = df[df.duplicated(subse
                .sort_values(['客户ID',
                .reset_index()
print(duplicates)
```

	index	交易ID	客户ID	交易金额	交易时间
0	0	TXN001	C101	150.0	2023-10-01 10:00:00
1	3	TXN004	C101	150.0	2023-10-01 10:00:00
2	7	TXN008	C101	150.0	2023-10-01 10:00:00
3	13	TXN014	C101	150.0	2023-10-01 10:00:00
4	19	TXN020	C101	150.0	2023-10-01 10:00:00
5	1	TXN002	C102	200.0	2023-10-01 10:05:00
6	5	TXN006	C102	200.0	2023-10-01 10:05:00
7	11	TXN012	C102	200.0	2023-10-01 10:05:00
8	16	TXN017	C102	200.0	2023-10-01 10:05:00
9	2	TXN003	C103	99.9	2023-10-01 10:10:00
10	8	TXN009	C103	99.9	2023-10-01 10:10:00
11	17	TXN018	C103	99.9	2023-10-01 10:10:00

（2）删除重复行

DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)

- subset：判断是否重复的列集合，默认是全部列。
- keep：特定字符串，表示重复时**保留**哪个记录数据。first表示保留第一条，last表示保留最后一条，False表示有重复的都不保留，**默认为first。**
- inplace：一个布尔值，表示是否在原数据上进行操作，默认为False，**返回去重后的新数据帧**，原数据帧不发生改变；当inplace为True时，函数duplicates**返回None**，原DataFrame的数据发生修改。

[示例] 不使用drop_duplicates删除全重复行

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('--------------------')
b = df.duplicated()
print(b)
print('--------------------')
c = df[b==False]
print(c)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
--------------------
0    False
1    False
2     True
3    False
4    False
5     True
dtype: bool
--------------------
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
```

[示例] 使用drop_duplicates删除全列重复行

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('----------------------')
df1 = df.drop_duplicates()
print(df1)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
```

[示例] 在原数据帧中删除指定列重复的行

```
import pandas as pd
df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('----------------------')
# 原地删除brand重复的行
df.drop_duplicates(subset='brand', inplace=True)
print(df)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
4     c  pack     5.0
```

[示例] 从后往前删除指定列重复的行

```
import pandas as pd

df = pd.DataFrame({
    'brand': ['a','b', 'a', 'a', 'c', 'c'],
    'style': ['cup','cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 4, 3.5, 5, 5]})
print(df)
print('----------------------')
# 根据band判断是否重复，且保留最后一条重复数据
df1 = df.drop_duplicates(subset='brand',keep='last')
print(df1)
```

```
  brand style  rating
0     a   cup     4.0
1     b   cup     4.0
2     a   cup     4.0
3     a   cup     3.5
4     c  pack     5.0
5     c  pack     5.0
----------------------
brand style  rating
1     b   cup     4.0
3     a   cup     3.5
5     c  pack     5.0
```

[示例] 删除重复列

```
import pandas as pd

df = pd.DataFrame({
    'style': ['a','b', 'a', 'a'],
    'type': ['a','b', 'a', 'a'],
    'rating': [4, 4, 4, 3.5]})
print(df)
print('----------------------')
print(df.T)
print('----------------------')
df = df.T.drop_duplicates().T
print(df)
```

```
  style type  rating
0     a    a     4.0
1     b    b     4.0
2     a    a     4.0
3     a    a     3.5
----------------------
           0  1  2    3
style      a  b  a    a
type       a  b  a    a
rating   4.0  4.0  4.0  3.5
----------------------
  style rating
0     a    4.0
1     b    4.0
2     a    4.0
3     a    3.5
```

[应用实例] 使用drop_duplicates()函数删除支付系统重试导致的重复交易。

```
# 保留每组第一条记录
cleaned_df = df.drop_duplicates(subset=['客户ID', '交易时间'], keep='first')
                               .reset_index()
print(cleaned_df)
```

	index	交易ID	客户ID	交易金额	交易时间
0	0	TXN001	C101	150.0	2023-10-01 10:00:00
1	1	TXN002	C102	200.0	2023-10-01 10:05:00
2	2	TXN003	C103	99.9	2023-10-01 10:10:00
3	4	TXN005	C104	300.0	2023-10-01 10:15:00
4	6	TXN007	C105	180.5	2023-10-01 10:20:00
5	9	TXN010	C106	210.0	2023-10-01 10:25:00
6	10	TXN011	C107	175.0	2023-10-01 10:30:00
7	12	TXN013	C108	225.0	2023-10-01 10:35:00
8	14	TXN015	C109	190.0	2023-10-01 10:40:00
9	15	TXN016	C110	240.0	2023-10-01 10:45:00
10	18	TXN019	C111	260.0	2023-10-01 10:50:00

4.1.2 缺失值处理

（1）检测缺失值

在处理缺失值前，需要先找到缺失值，使用人工查找缺失值，效率低且容易遗漏。函数isnull可以检查数据中的缺失值，返回一个布尔值的矩阵，每一个布尔值表示对应位置的数据是否缺失。

函数notnull与isnull意思相反。返回的布尔值为True时表示非缺失值。

[示例] 检测缺失值

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'a':[1,2,nan,nan],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                  'd':[13,14,15,16]})

print(df)
print('--------------------------')
print(df.isnull())
print('--------------------------')
print(df.notnull())
```

```
     a    b     c     d
0  1.0  5.0   9.0    13
1  2.0  NaN  10.0  14
2  NaN  7.0  11.0   15
3  NaN  8.0   NaN  16
--------------------------
       a      b      c      d
0  False  False  False  False
1  False   True  False  False
2   True  False  False  False
3   True  False   True  False
--------------------------
       a      b      c      d
0   True   True   True   True
1   True  False   True   True
2  False   True   True   True
3  False   True  False   True
```

[示例] 检测缺失值数目大于1的列并删除。

```
import pandas as pd
from numpy import nan
df = pd.DataFrame({'a':[1,2,nan,nan],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                  'd':[13,14,15,16]})
print(df)
print('--------------------------')
csum = df.isnull().sum()   # 计算时True为1、False为0
print('--------------------------')
print(csum)
print('--------------------------')
idx = csum[csum>1].index
print('--------------------------')
ndf = df.drop(columns=idx)
print(ndf)
```

```
     a    b     c    d
0  1.0  5.0   9.0   13
1  2.0  NaN  10.0   14
2  NaN  7.0  11.0   15
3  NaN  8.0   NaN   16
--------------------------
a    2
b    1
c    1
d    0
dtype: int64
--------------------------
     b     c     d
0  5.0   9.0   13
1  NaN  10.0   14
2  7.0  11.0   15
3  8.0   NaN   16
```

[示例] 检测缺失值数目>=50%的列并删除。

```
import pandas as pd
from numpy import nan
df = pd.DataFrame({'a':[1,2,nan,nan],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                  'd':[13,14,15,16]})
print(df)
print('--------------------------')
csum = df.isnull().mean() # 非空值数目/所有值数目
print('--------------------------')
print(csum)
print('--------------------------')
idx = csum[csum>=0.5].index
print('--------------------------')
ndf = df.drop(columns=idx)
print(ndf)
```

```
     a      b      c      d
0  False  False  False  False
1  False   True  False  False
2   True  False  False  False
3   True  False   True  False
--------------------------
a    0.50
b    0.25
c    0.25
d    0.00
dtype: float64
--------------------------
     b     c     d
0  5.0   9.0   13
1  NaN  10.0   14
2  7.0  11.0   15
3  8.0   NaN   16
```

（2）删除缺失值

DataFrame.dropna(axis=0, **how='any',** thresh=None, subset=None, inplace=False)

- axis=0，**默认删除包含NaN的行**，axis=1则删除包含NaN的列。
- how：有两个取值：any（默认）表示如果存在任何NaN值就删除，all表示如果所有的值均为NaN值才删除。
- thresh：thresh=N表示保留至少有N个非NA的行（列）
- subset：删除subset指定的列有NaN的行
- inplace：默认值 False，在副本上完成删除。若设为True，则原数据被修改。

[示例] 删除包含缺失值的行

```
import pandas as pd
from numpy import nan as NA

df = pd.DataFrame({'a':[NA,NA,NA,NA],
                   'b':[5,NA,NA,8],
                   'c':[9,10,11,NA],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)

df = df.dropna()
print('删除包含NaN值的行后：')
print(df)
```

```
原数据：
    a    b     c     d
0 NaN  5.0   9.0  13
1 NaN  NaN  10.0  14
2 NaN  NaN  11.0  15
3 NaN  8.0   NaN  16

删除包含NaN值的行后：
Empty DataFrame
Columns: [a, b, c, d]
Index: []
```

[示例] 删除a列有NaN的行

```
import pandas as pd
from numpy import nan as NA

df = pd.DataFrame({'a':[9,NA,NA,NA],
                   'b':[5,NA,NA,8],
                   'c':[9,10,11,NA],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)

df = df.dropna(subset='a')
print('删除a列有缺失的行后：')
print(df)
```

原数据：

```
    a    b     c   d
0  9.0  5.0   9.0  13
1  NaN  NaN  10.0  14
2  NaN  NaN  11.0  15
3  NaN  8.0   NaN  16
```

删除包含NaN值的行后：

```
    a    b    c   d
0  9.0  5.0  9.0  13
```

[示例] 删除包含缺失值的列

```
import pandas as pd
from numpy import nan as NA

df = pd.DataFrame({'a':[NA,NA,NA,NA],
                   'b':[5,NA,NA,8],
                   'c':[9,10,11,NA],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)

df = df.dropna(axis=1)
print('删除包含NaN值的列后：')
print(df)
```

原数据：

```
    a    b     c   d
0 NaN  5.0   9.0  13
1 NaN  NaN  10.0  14
2 NaN  NaN  11.0  15
3 NaN  8.0   NaN  16
```

删除包含NaN值的列后：

```
    d
0  13
1  14
2  15
3  16
```

[示例] 删除全都是NaN的列

```
import pandas as pd
from numpy import nan as NA

df = pd.DataFrame({'a':[NA,NA,NA,NA],
                   'b':[5,NA,NA,8],
                   'c':[9,10,11,NA],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)

df = df.dropna(axis=1, how='all')
print('删除全部为NaN值的列后：')
print(df)
```

原数据：

```
    a    b     c    d
0 NaN  5.0   9.0   13
1 NaN  NaN  10.0  14
2 NaN  NaN  11.0  15
3 NaN  8.0   NaN  16
```

删除包含NaN值的行后：

```
    b     c    d
0  5.0   9.0   13
1  NaN  10.0  14
2  NaN  11.0  15
3  8.0   NaN  16
```

[示例] 保留>行数的50%个非NaN值的列

```
import pandas as pd
from numpy import nan as NA

df = pd.DataFrame({'a':[9,NA,NA,NA],
                   'b':[5,NA,NA,8],
                   'c':[9,10,11,NA],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)

df = df.dropna(axis=1, thresh=int(len(df)*0.5))
print('过滤掉a列有缺失的行后：')
print(df)
```

原数据：

	a	b	c	d
0	9.0	5.0	9.0	13
1	NaN	NaN	10.0	14
2	NaN	NaN	11.0	15
3	NaN	8.0	NaN	16

过滤掉a列有缺失的行后：

	b	c	d
0	5.0	9.0	13
1	NaN	10.0	14
2	NaN	11.0	15
3	8.0	NaN	16

（3）替换缺失值

replace(self, to_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad', axis=None)

- to_replace：参数to_replace是序列中原有的值（不一定是缺失值），查找到该值之后，把该值替换为参数value指定的值
- limit：替换的最大次数
- regex：是否把to_replace 解释为正则表达式，默认值是False。如果设置为True，那么参数to_replace必须是字符串
- method：有效值是pad、ffill、bfill，当参数to_replace是标量、列表或字典，并且参数value是None时，使用method方法来替换。

[示例] 用指定值替换缺失值

```
import pandas as pd
from numpy import nan,mean

df = pd.DataFrame({'a':[1,2,nan,4],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                   'd':[13,14,15,16]})
print(df)
print('-------------------------')
# 用0替换NaN值
print(df.replace(nan,0))
print('-------------------------')
# 用每一列的均值替换NaN值
print(df.replace(nan, df.mean().round(2)))
```

```
   a    b    c     d
0  1.0  5.0   9.0   13
1  2.0  NaN  10.0  14
2  NaN  7.0  11.0  15
3  4.0  8.0   NaN  16
-------------------------
   a    b    c    d
0  1.0  5.0   9.0  13
1  2.0  0.0  10.0  14
2  0.0  7.0  11.0  15
3  4.0  8.0   0.0  16
-------------------------
    a     b     c   d
0  1.00  5.00   9.0  13
1  2.00  6.67  10.0  14
2  2.33  7.00  11.0  15
3  4.00  8.00  10.0  16
```

[示例] 多值对应替换

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'a':[1,2,nan,0],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                   'd':[13,14,0,16]})
print('原数据：')
print(df)
print('-------------------------')

# 将0替换成1，nan替换成0
print(df.replace({0:1,nan:0}))
#或print(df.replace([0,nan],[1,0]))
```

原数据：

```
     a    b     c     d
0  1.0  5.0   9.0    13
1  2.0  NaN  10.0    14
2  NaN  7.0  11.0     0
3  0.0  8.0   NaN    16
-------------------------
     a    b     c   d
0  1.0  5.0   9.0  13
1  2.0  0.0  10.0  14
2  0.0  7.0  11.0   1
3  1.0  8.0   0.0  16
```

[示例] 多值统一替换

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'a':[1,2,nan,0],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                   'd':[13,14,0,16]})
print('原数据：')
print(df)
print('-------------------------')

# 将0和nan都替换成1
print(df.replace([0,nan], 1))
```

原数据：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	NaN	7.0	11.0	0
3	0.0	8.0	NaN	16

---

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	1.0	10.0	14
2	1.0	7.0	11.0	1
3	1.0	8.0	1.0	16

[示例] 指定方法替换

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'a':[1,2,nan,0],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                   'd':[13,14,0,16]})
print('原数据：')
print(df)
print('-------------------------')

# 将nan替换成其后一个元素的值
print(df.replace(nan,method='bfill'))
```

原数据：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	NaN	7.0	11.0	0
3	0.0	8.0	NaN	16

---

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	7.0	10.0	14
2	0.0	7.0	11.0	0
3	0.0	8.0	NaN	16

[示例] 在原数据上替换指定列的NaN

```
import pandas as pd
from numpy import nan
df = pd.DataFrame({'a':[1,2,nan,0],
                   'b':[5,nan,7,8],
                   'c':[9,10,11,nan],
                   'd':[13,14,0,16]})
print('原数据：')
print(df)

print('a列NaN被替换后：')
df['a'].replace(nan, -1, inplace=True)
print(df)
```

原数据：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	NaN	7.0	11.0	0
3	0.0	8.0	NaN	16

a列NaN被替换后：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	-1.0	7.0	11.0	0
3	0.0	8.0	NaN	16

[示例] 利用正则表达式替换df中的数据

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'a':[1,2,nan,4],
            'b':[5,nan,7,8], 'c':[9,10,11,nan], 'd':[13,14,15,16],
            'e':['2024-02.26','2024-03.04','2024-03.11','2024-03.18']})
print(df)

df.replace('[-.]', '/', regex=True,inplace=True)
print(df)
```

	a	b	c	d	e
0	1.0	5.0	9.0	13	2024-02.26
1	2.0	NaN	10.0	14	2024-03.04
2	NaN	7.0	11.0	15	2024-03.11
3	4.0	8.0	NaN	16	2024-03.18

	a	b	c	d	e
0	1.0	5.0	9.0	13	2024/02/26
1	2.0	NaN	10.0	14	2024/03/04
2	NaN	7.0	11.0	15	2024/03/11
3	4.0	8.0	NaN	16	2024/03/18

（4）填充缺失值

fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, **kwargs)

- value：用于填充的空值的值。
- method： {'backfill', 'bfill', 'pad', 'ffill', None}, default None。定义了填充空值的方法， pad / ffill表示用前面行/列的值填充当前行/列的空值，backfill / bfill表示用后面行/列的值填充当前行/列的空值。
- axis：轴。0或'index'，表示按行填充；1或'columns'，表示按列填充。
- inplace：是否原地替换。

[示例] 用0填充缺失值

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'a':[1,2,np.nan,4],
                   'b':[5,np.nan,7,8],
                   'c':[9,10,11,np.nan],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)
print('-------------------------')
# 用0填充nan
print(df.fillna(0))
```

原数据：

```
     a    b     c     d
0  1.0  5.0   9.0    13
1  2.0  NaN  10.0  14
2  NaN  7.0  11.0  15
3   4.0  8.0   NaN  16
-------------------------
    a    b     c   d
0  1.0  5.0   9.0  13
1  2.0  0.0  10.0  14
2  0.0  7.0  11.0  15
3  4.0  8.0   0.0  16
```

[示例] 用每一列的平均值填充缺失值

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'a':[1,2,np.nan,4],
                   'b':[5,np.nan,7,8],
                   'c':[9,10,11,np.nan],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)
print('-------------------------')
print(df.fillna(df.mean()))
```

原数据：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	NaN	7.0	11.0	15
3	4.0	8.0	NaN	16

---

	a	b	c	d
0	1.000000	5.000000	9.0	13
1	2.000000	6.666667	10.0	14
2	2.333333	7.000000	11.0	15
3	4.000000	8.000000	10.0	16

[示例] 用后一个值填充列缺失值

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'a':[1,2,np.nan,4],
                   'b':[5,np.nan,7,8],
                   'c':[9,10,11,np.nan],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)
print('-------------------------')
print(df.fillna(method='bfill'))
```

原数据：

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	NaN	10.0	14
2	NaN	7.0	11.0	15
3	4.0	8.0	NaN	16

---

	a	b	c	d
0	1.0	5.0	9.0	13
1	2.0	7.0	10.0	14
2	4.0	7.0	11.0	15
3	4.0	8.0	NaN	16

[示例] 用后一个值填充行缺失值

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'a':[1,2,np.nan,4],
                   'b':[5,np.nan,7,8],
                   'c':[9,10,11,np.nan],
                   'd':[13,14,15,16]})
print('原数据：')
print(df)
print('-------------------------')
print(df.fillna(method='bfill',axis=1))
```

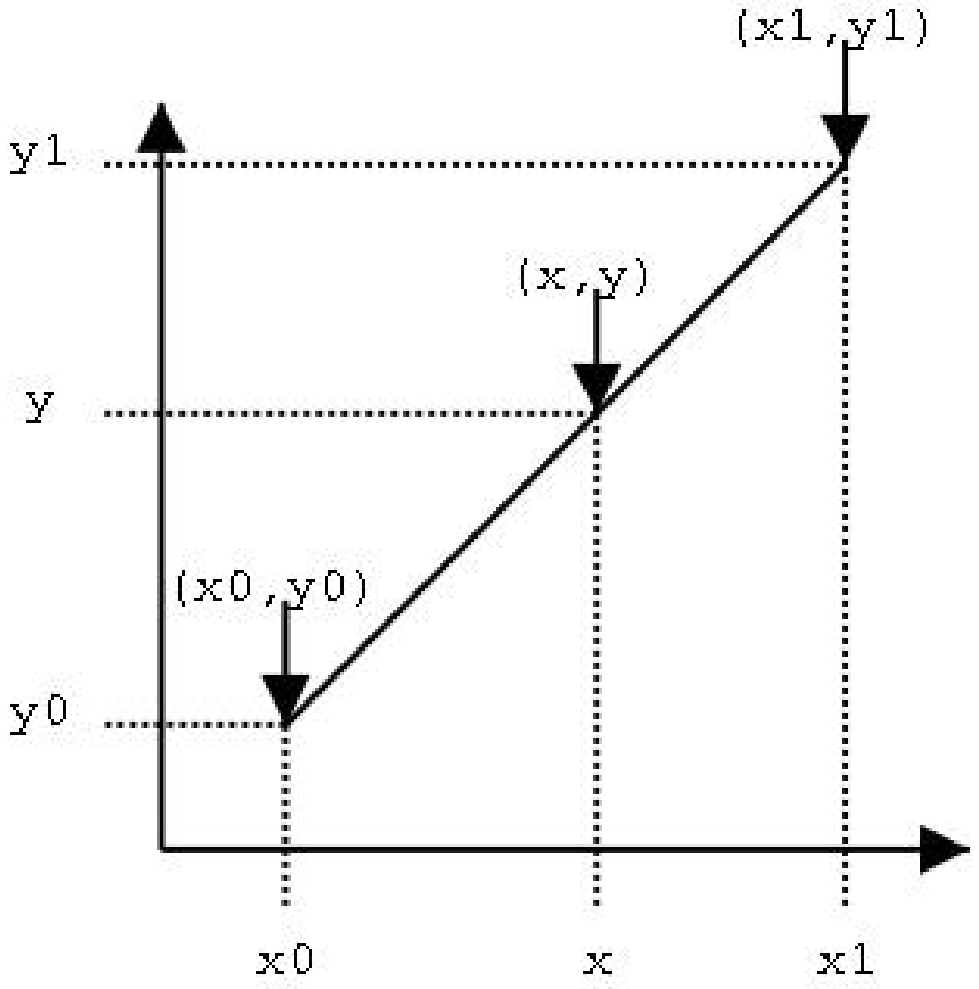
原数据：

```
    a   b    c   d
0  1.0  5.0  9.0  13
1  2.0  NaN  10.0  14
2  NaN  7.0  11.0  15
3  4.0  8.0  NaN  16
-------------------------
    a    b    c    d
0  1.0  5.0  9.0  13.0
1  2.0  10.0  10.0  14.0
2  7.0  7.0  11.0  15.0
3  4.0  8.0  16.0  16.0
```

（5）插值法填充缺失值

- 除了上述的填充方法外，还有一种效果更好的方法——插值法。
- 插值，是根据已知的数据序列，找到其中的规律，然后根据这个规律，来对其中尚未有数据记录的点进行**数值估计**。
- 插值法有三种常用的方法：线性插值、多项式插值和样条插值.
  - **线性插值**根据已知数值构建线性方程组，通过求解线性方程组获得缺失值；
  - **多项式插值**是通过拟合多项式，通过多项式求解缺失值，多项式插值中最常用的是拉格朗日插值法和牛顿插值法；
  - **样条插值**是通过可变样条做出一条经过一系列点的光滑曲线的插值方法。

线性插值



$$\frac{y-y_0}{x-x_0}=\frac{y_1-y_0}{x_1-x_0}$$

x = [1, 2, 3, 4, 5, 6, 9, 10, 11, 12]
y = [10,16, 21, 32, 35, 43, 58, 62, 67,70]

用线性插值方法得到x=7对应的y值：

$$\frac{y-43}{7-6}=\frac{58-43}{9-6}$$

得到y=48

x = [1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12]
y = [10,16, 21, 32, 35, 43, 48, 58, 62, 67,70]

接着计算x=8对应的y值：

$$\frac{y-48}{8-7}=\frac{58-48}{9-7}$$

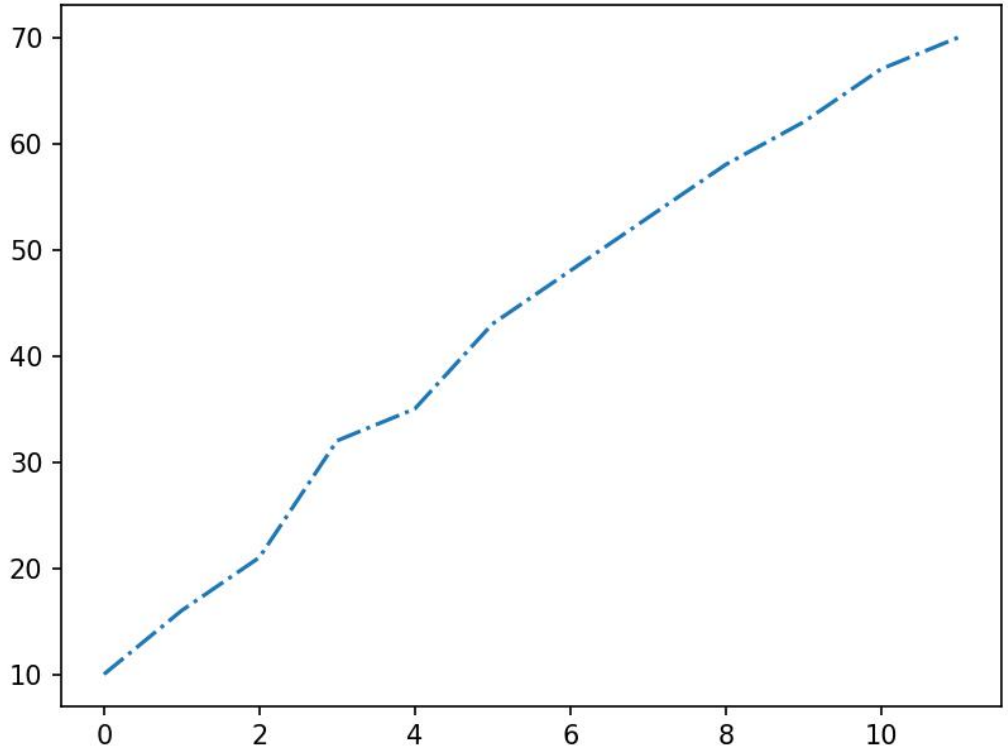
得到y=53

[示例] 一维线性插值

```
import scipy.interpolate as interpolate
import numpy as np
import matplotlib.pyplot as plt

a = [ 1,  2,  3,  4,  5,  6,  9, 10, 11,12]
b = [10,16,21,32,35,43,58, 62,67,70]


# 1维线性插值函数
s = interpolate.interp1d(a,b,kind='linear')
x = range(1,13)
y = s(x)
plt.plot( x, y,  '-.' )
print('线性插值法求出的s[7,8]=', s([7,8]))
plt.show()
```



线性插值法求出的s[7,8]= [48. 53.]

多项式插值

给定$n+1$个点$\{(x_i, y_i)\}_{i=0}^{n}$（称为 ***插值点***），所谓 ***多项式插值*** 就是找到一个多项式（称为 ***插值多项式***）

$$y = P(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$$

使得它满足条件

$$y_i = P(x_i)\text{，其中} i = 0, 1, \ldots, n$$

也就是说，多项式$y = P(x)$的图像要经过给定的$n+1$个点。

常见的多项式插值法有**拉格朗日插值法、牛顿插值法**等。

拉格朗日插值

拉格朗日基本多项式:

$$\ell_j(x) := \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i}$$

x = [1, 2, 3, 4, 5, 6, 9, 10, 11, 12]
y = [10,16, 21, 32, 35, 43, 58, 62, 67,70]

例：已知三个点(4,32),(5,35),(6,43),
用拉格朗日插值求x=7, y=?

$$\boldsymbol{l_1(x) = \frac{(7-5)(7-6)}{(4-5)(4-6)} = 1}$$

$$\boldsymbol{l_2(x) = \frac{(7-4)(7-6)}{(5-4)(5-6)} = -3}$$

$$\boldsymbol{l_3(x) = \frac{(7-4)(7-5)}{(6-4)(6-5)} = 3}$$

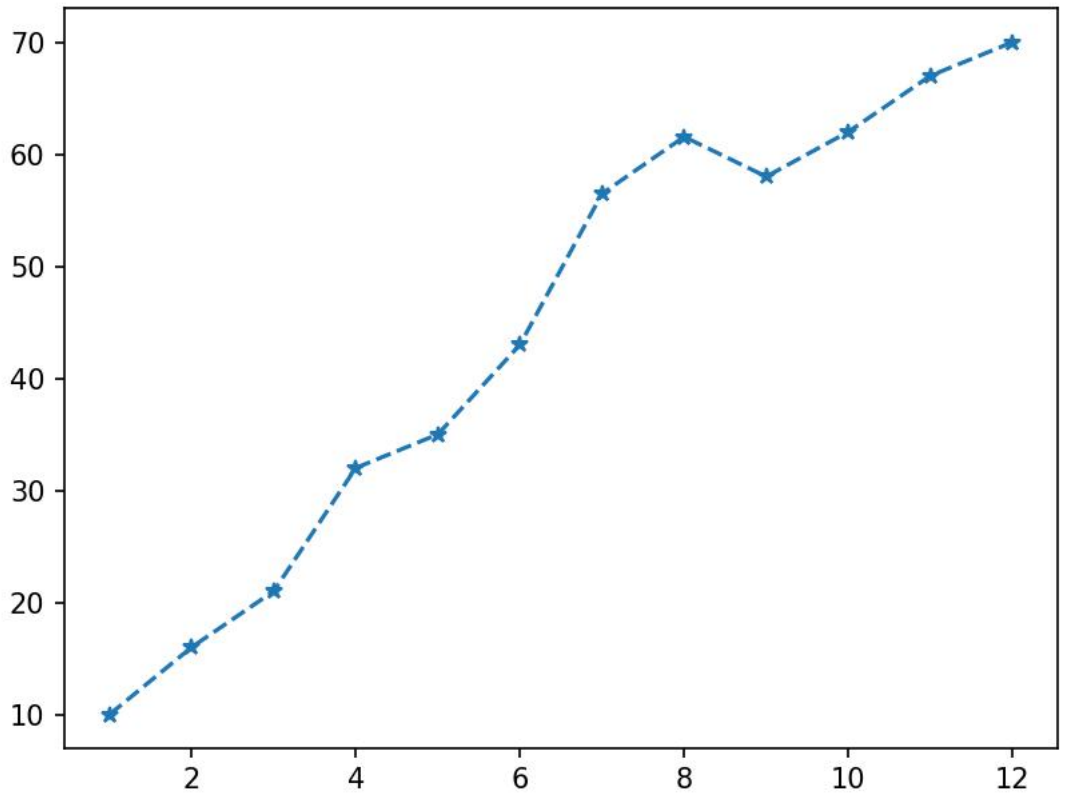
$$\boldsymbol{y = 32 * 1 + 35 * (-3) + 43 * 3 = 56}$$

[示例] 拉格朗日插值

```
import scipy.interpolate as interpolate
import numpy as np
import matplotlib.pyplot as plt

a = [ 1,  2,  3,  4,  5,  6,  9, 10, 11,12]
b = [10,16,21,32,35,43,58, 62,67,70]

la = interpolate.lagrange(a,b)
print(la([7,8]))
```




```
plt.plot(np.arange(1,13),la(np.arange(1,13)),'--',marker='*')
plt.show()
```

```
[56.48051948 61.55757576]
```

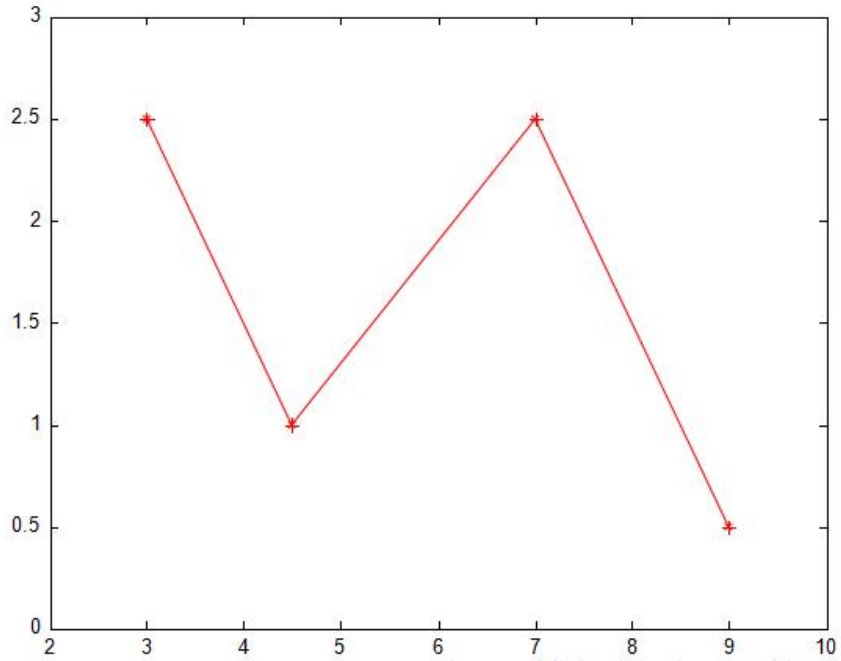
样条插值

- 每两个点之间确定一个函数，这个函数就是一个样条，然后把所有样条分段结合成一个函数，就是最终的插值函数。
- 样条插值是一种工业设计中常用的、得到平滑曲线的一种插值方法，三次样条又是其中用的较为广泛的一种。

（1）线性样条

两点确定一条直线，我们可以在每两点间画一条直线，就可以把所有点连起来。

显然曲线不够光滑，因为连接点处导数不相同。



http://blog.csdn.net/deramer1

（2）二次样条

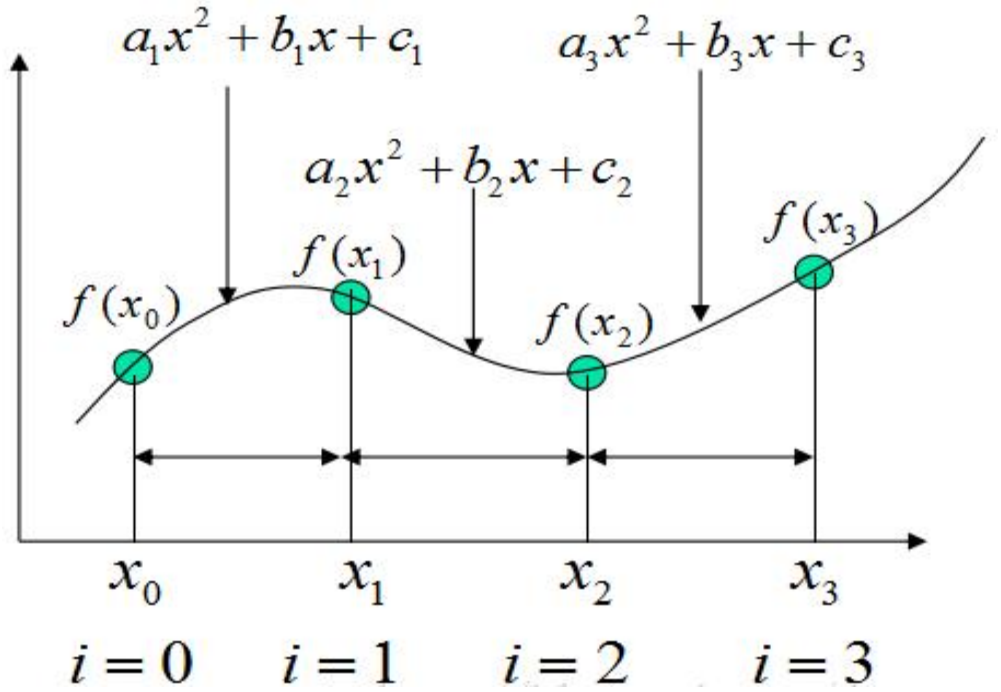
用曲线代替直线，二次函数是最简单的曲线。
假设4个点，x_0，x_1，x_2，x_3，有3个区间，需要3个二次样条，每个二次样条为 ax^2+bx+c，总计9个未知系数。

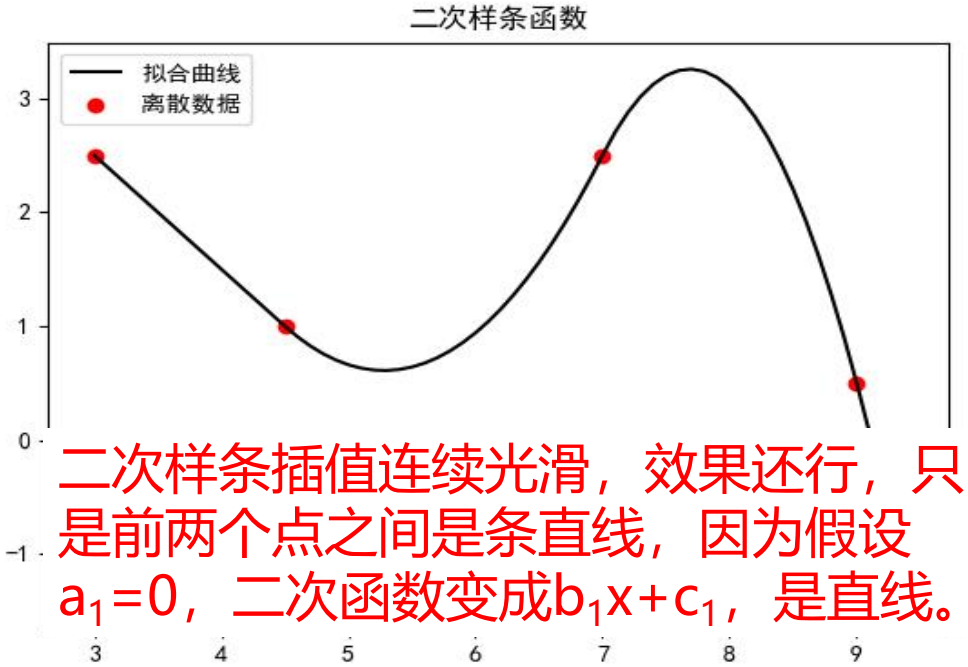
- 两个端点x_0、x_3都有一个二次函数经过，可确定2个方程，如$ax_0^2+bx_0+c = f(x_0)$
- 两个中间点x_1、x_2都有两个二次函数经过，可确定4个方程。
- 中间点处必须连续，需要保证左右二次函数一阶导相等，可确定2个方程：

$$2*a_1*x_1 + b_1 = 2*a_2*x_1 + b_2$$
$$2*a_2*x_2 + b_2 = 2*a_3*x_2 + b_3$$

- 这里假设第一方程的二阶导为0，即 $a_1=0$，又是一个方程，共计9个方程。

联立即可求解。





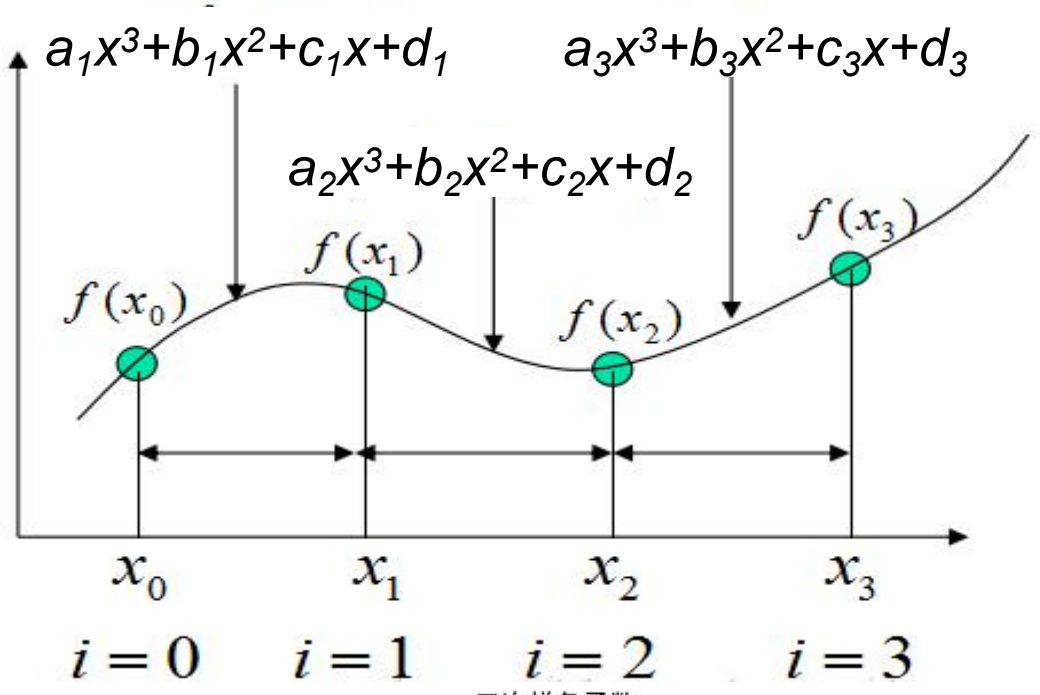
二次样条插值连续光滑，效果还行，只是前两个点之间是条直线，因为假设$a_1=0$，二次函数变成b_1x+c_1，是直线。

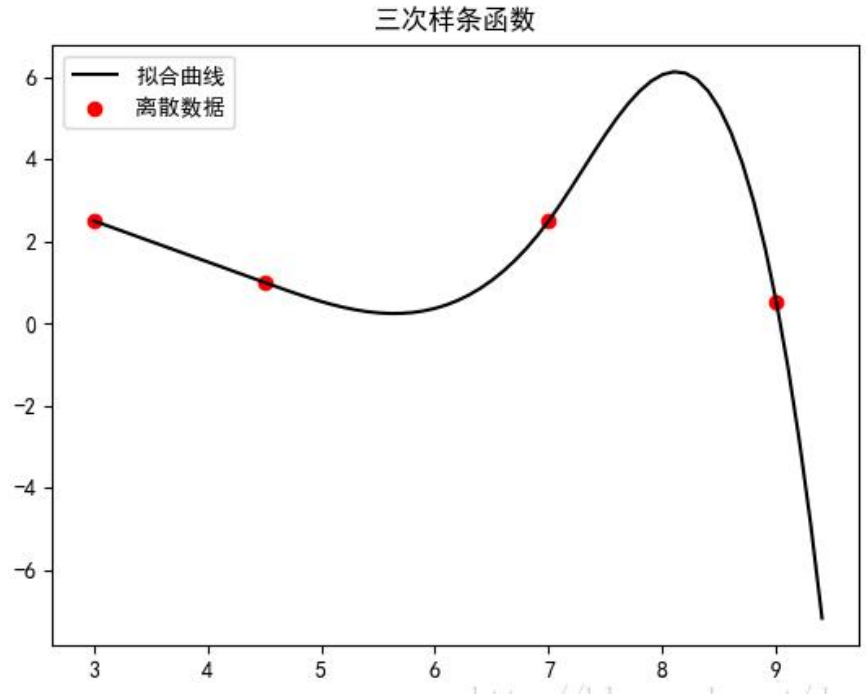
（2）三次样条

二次函数最高项系数为0，导致变成直线，那三次函数最高项系数为0，还是曲线，插值效果更好。同样假设4个点，x_0，x_1，x_2，x_3，有3个区间，需要3个三次样条，每个三次样条为 ax^3+bx^2+cx+d，故总计12个未知数。

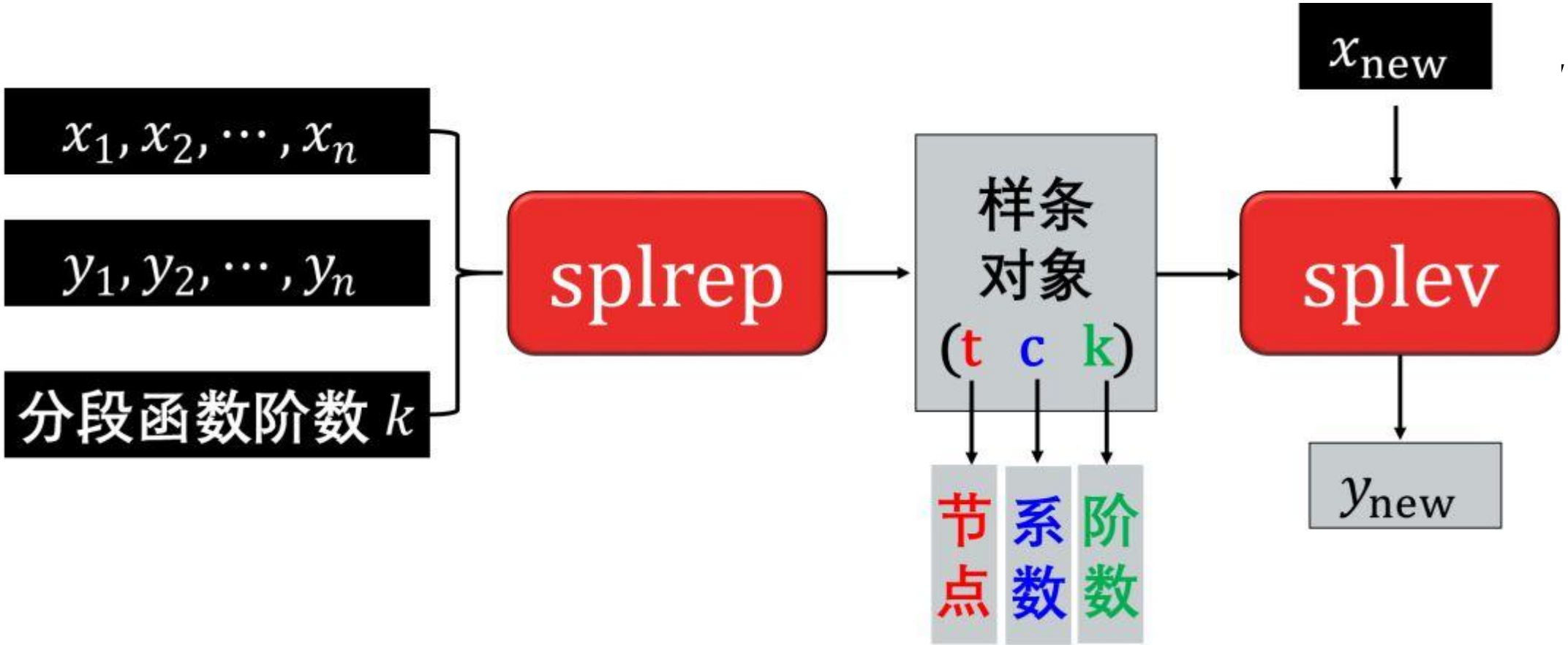
- 4个节点坐标代入三次样条函数得到4个方程： $y_i = ax_i^3+bx_i^2+cx_i+d$
- 两个函数在节点处的函数值相等，得到2个方程；
- 两个函数在节点处的一阶导数应该相等（节点处光滑），得到两个方程。
- 两个函数在节点处的二阶导数应该相等，得到两个方程。
- 假设端点处的二阶导数为零，这里是两个方程： $a_1=0$

  $b_1=0$





- 插值使用的函数有splrep（spline representation）函数和splev（spline evaluation）函数，前者将 x, y 和插值方式转换成样条对象tck，后者利用tck在 xnew 上生成 ynew：



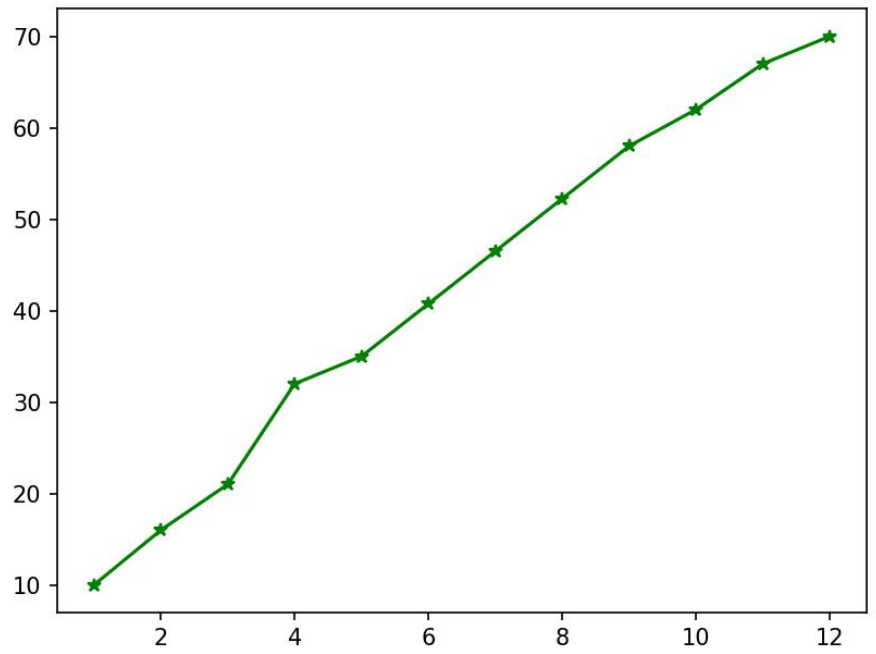
[示例] 线性样条插值法

```
from scipy.interpolate import splrep, splev
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x = np.array([1, 2, 3, 4, 5, 9, 10, 11, 12])
y = np.array([10, 16, 21, 32, 35, 58, 62, 67, 70])
x_new = np.arange(1,13)  # 插值点

tck_linear = splrep(x, y, k=1) # 生成样条表示 (k=1表示线性)
y_linear = splev(x_new, tck_linear)   # 计算插值

print("\n线性样条在x=7的预测:%.1f"%splev(7, tck_linear))
print("线性样条在x=8的预测:%.1f"%splev(8, tck_linear))
plt.plot(x_new, y_linear,c='g',marker='*')
plt.show()
```



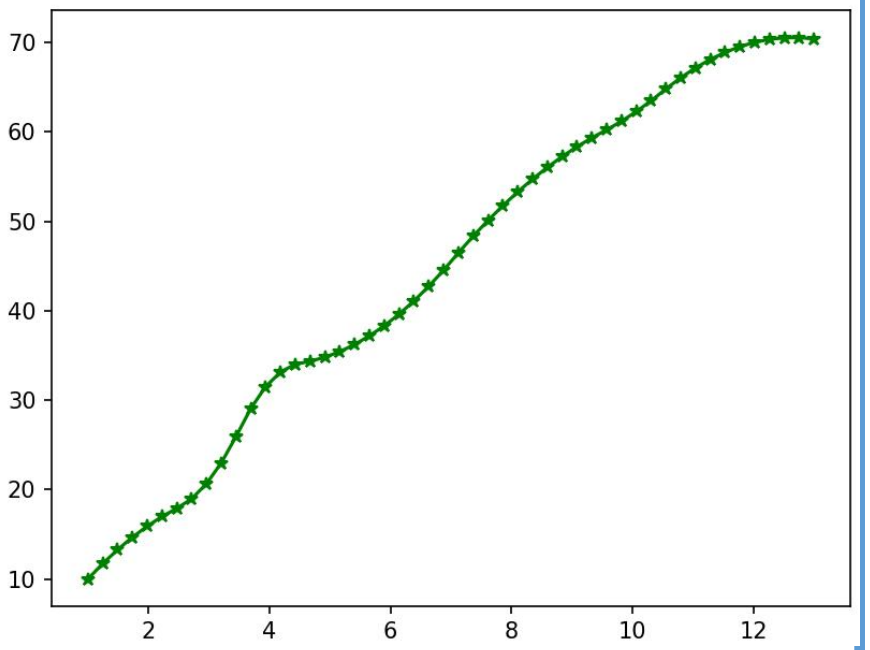

```
线性样条在x=7的预测:46.5
线性样条在x=8的预测:52.2
```

[示例] 二次样条插值法

```
from scipy.interpolate import splrep, splev
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x = np.array([1, 2, 3, 4, 5, 9, 10, 11, 12])
y = np.array([10, 16, 21, 32, 35, 58, 62, 67, 70])

x_new = np.linspace(1,13,50)  # 50个插值点
tck_quad = splrep(x, y, k=2)  # 生成二次样条表示
y_quad = splev(x_new, tck_quad) # 计算插值
print("\n二次样条在x=7的预测:%.1f"%splev(7, tck_quad))
print("二次样条在x=8的预测:%.1f"%splev(8, tck_quad))
plt.plot(x_new, y_quad,c='g',marker='*')
plt.show()
```



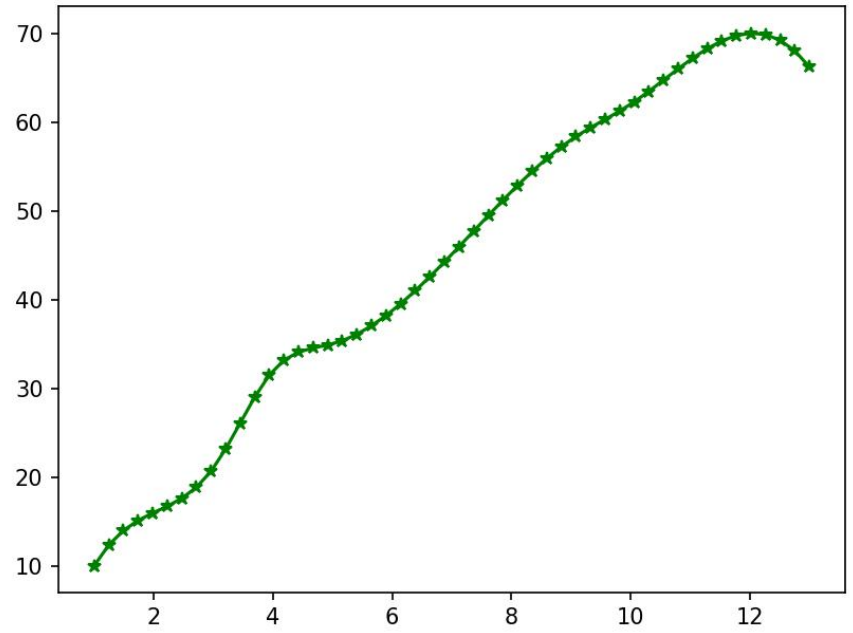

```
二次样条在x=7的预测:45.5
二次样条在x=8的预测:52.7
```

[示例] 三次样条插值法

```
from scipy.interpolate import splrep, splev
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x = np.array([1, 2, 3, 4, 5, 9, 10, 11, 12])
y = np.array([10, 16, 21, 32, 35, 58, 62, 67, 70])

x_new = np.linspace(1,13,50)  # 50个插值点
tck_cubic = splrep(x, y, k=3)
y_cubic = splev(x_new, tck_cubic)
print("\n三次样条在x=7的预测:%.1f"%splev(7, tck_cubic))
print("三次样条在x=8的预测:%.1f"%splev(8, tck_cubic))
plt.plot(x_new, y_cubic,c='g',marker='*')
plt.show()
```



三次样条在x=7的预测:45.1
三次样条在x=8的预测:52.2

5.1.3 异常值的处理

原始数据中可能会出现明显违背自然规律的数据，这些数据属于噪音，对数据分析会造成很大的干扰，对分析结果具有巨大的不良影响。

（1）检测异常值

异常值的检测**通常采用绘制图形**，从图形观察数据分布情况，找出离群点。

离群点可能是异常值，但是不绝对，有些时候离群点的数据也可能是正常的。异常值的判断需要有行业背景和业务知识。

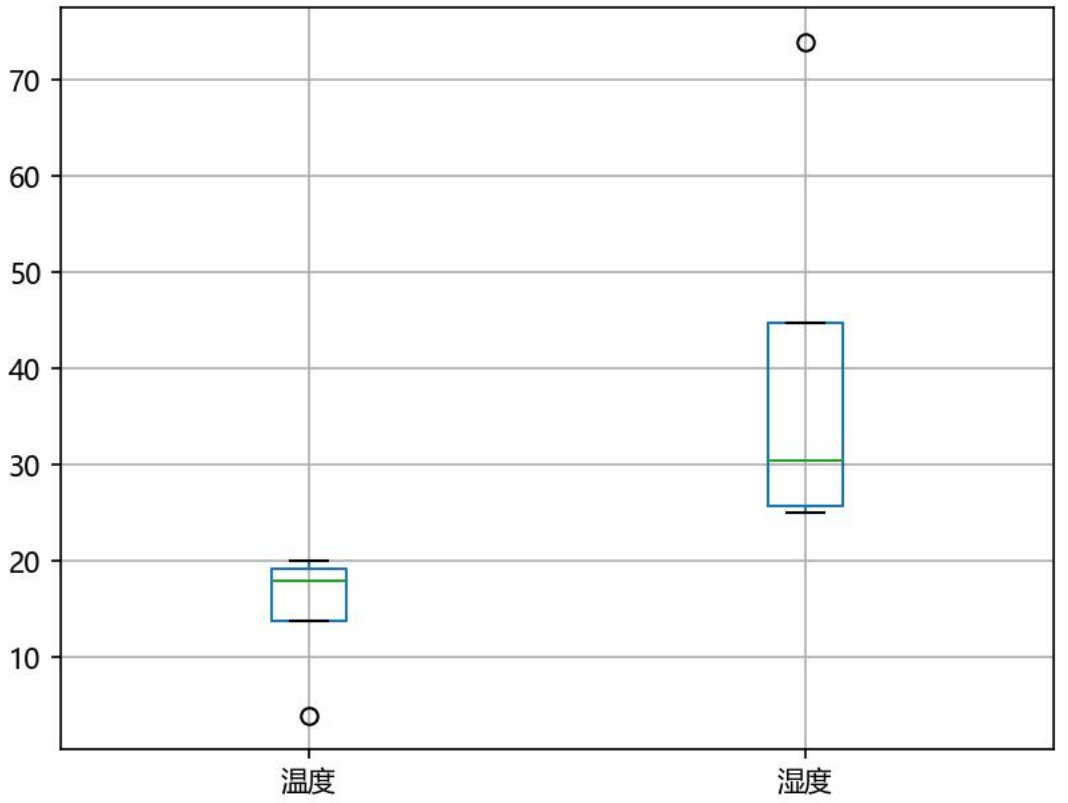
[示例] 检测异常值

箱线图实际就是利用数据的分位数识别其中的异常点。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(
                {'温度':[17,20,19,4],
                 '湿度':[35,26,74,25],
                })

plt.rcParams['font.sans-serif'] = [Simhei']
df.boxplot()
plt.show()
```



（2）处理异常值

异常值处理主要有如下三种方法。

（1）删除。删除包含有异常值的记录。

（2）视为缺失值。将异常值视为缺失值，利用缺失值的处理方法进行处理。

（3）平均值修正。使用前后两个值的平均值或是整列的数据平均值修正异常值。

【应用实例】对二手房数据进行清洗

```
# (1) 读取数据，查看数据摘要信息
import pandas as pd
data = pd.read_excel('e:/handroom.xlsx')
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1058 entries, 0 to 1057
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   区          1058 non-null   object
 1   小区名称       1057 non-null   object
 2   标题         1058 non-null   object
 3   房屋信息       1058 non-null   object
 4   关注         1058 non-null   object
 5   地铁         441 non-null    object
 6   单价(元/平米)  1058 non-null   float64
dtypes: float64(1), object(6)
memory usage: 58.0+ KB
```

可以看出，二手房数据各列非空值的数量以及数据类型， 还能看到除了“地铁”列外，小区名称也有缺失。

【应用实例】对二手房数据进行清洗

```
# （2） 删除包含'小区名称'为NaN值的行
data = data.dropna(subset='小区名称')
print(data)
```

```
        区          小区名称  ...    地铁  单价(元/平米)
0      锦江        翡翠城四期  ...  近地铁     176036.0
1      锦江     时代豪庭一期  ...     NaN      26959.4
2      锦江        卓锦城六期  ...     NaN      22612.8
3      锦江        星城银座  ...  近地铁      18014.5
4      锦江        新莲新苑  ...     NaN      13513.5
...     ...             ...  ...     ...          ...
1053   锦江        锦东庭园  ...  近地铁      25663.7
1054   锦江   五世同堂街72号  ...  近地铁      14288.3
1055   锦江   城市博客VC时代  ...  近地铁      18205.2
1056   锦江     大地城市脉搏  ...  近地铁      25089.1
1057   锦江     俊发星雅俊园  ...     NaN       8911.2

[1057 rows x 7 columns]
```

可以看出，删除缺失值后，二手房数据剩余1057行，比之前减少了一行。

【应用实例】对二手房数据进行清洗

```
# （3）检测二手房数据中有多少重复值
print(data[data.duplicated() == True])
```

```
979  锦江    人居锦尚春天B区  ...   NaN     15938.6
980  锦江      蓝润锦江春天  ...   NaN     16212.2
981  锦江      铁狮门晶融汇  ...  近地铁    42319.1
982  锦江            朗御  ...  近地铁    31111.1
983  锦江        国润新锦江  ...   NaN     17430.1
984  锦江   绿地468公馆二期  ...  近地铁    22727.3
985  锦江        莲花逸都  ...  近地铁    21044.1
986  锦江      翡翠城五期  ...   NaN     30900.0
987  锦江      时代豪庭三期  ...   NaN     36811.2

[58 rows x 7 columns]
```

原数据重复项共有58个。

【应用实例】对二手房数据进行清洗

```
# （4）删除重复值
data = data.drop_duplicates()
print(data)
```

Squeezed text (61 lines).

	区	小区名称	...	地铁	单价(元/平米)
0	锦江	翡翠城四期	...	近地铁	176036.0
1	锦江	时代豪庭一期	...	NaN	26959.4
2	锦江	卓锦城六期	...	NaN	22612.8
3	锦江	星城银座	...	近地铁	18014.5
4	锦江	新莲新苑	...	NaN	13513.5
...	...	...	...	...	...
1053	锦江	锦东庭园	...	近地铁	25663.7
1054	锦江	五世同堂街72号	...	近地铁	14288.3
1055	锦江	城市博客VC时代	...	近地铁	18205.2
1056	锦江	大地城市脉搏	...	近地铁	25089.1
1057	锦江	俊发星雅俊园	...	NaN	8911.2

[999 rows x 7 columns]

1057-58=999

【应用实例】对二手房数据进行清洗

```
# （5）找出同一小区数据多于10条的记录
uv = data['小区名称'].value_counts()
uv = uv[uv>20]
print(uv)
```

```
翡翠城四期          21
鑫苑名家一期        20
华都美林湾          18
锦江城市花园二期    17
恒大都汇华庭        16
锦江城市花园三期    15
望江橡树林一期      14
鑫苑名家二期        13
俊发星雅俊园        13
钢管厂五区          12
东湖国际            11
比华利国际城一期    11
Name: 小区名称, dtype: int64
```

因为每个楼盘的地理位置、配套设施、以及开盘时间各不相同，所以每个小区的出售价格也不相同，无法直接对“单价”列进行异常值检测。

为了准确地检测异常值，应分别对每个小区进行异常值检测。

【应用实例】对二手房数据进行清洗

```
# （6）提取‘鑫苑名家一期’的数据
xq = data[data['小区名称'] == '鑫苑名家一期']
print(xq)
```

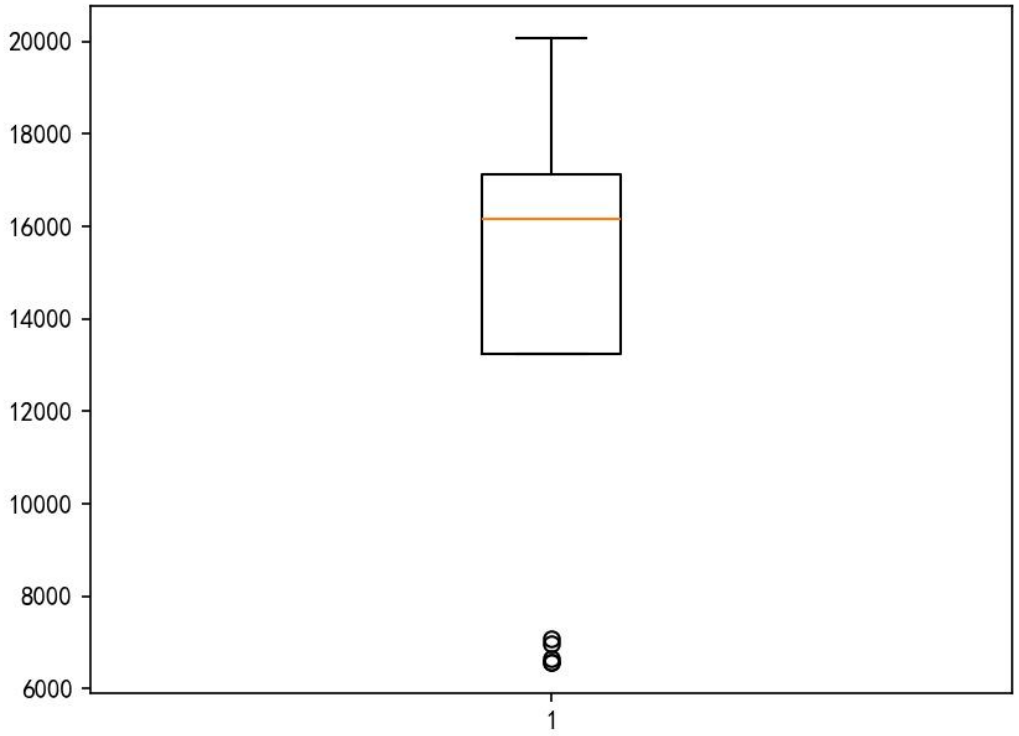
	区	小区名称	标题	房屋信息	关注	地铁	单价(元/平米)
150	锦江	鑫苑名家一期	鑫苑名家一期 3室1厅 东南	高楼层(共34层) | 2011年建 | 3室1厅 | 104平米 | 东南	22人关注 / 2月前发布	NaN	17115.4
213	锦江	鑫苑名家一期	鑫苑名家一期，套二带装修，中间楼层，	高楼层(共34层) | 2011年建 | 2室1厅 | 79平米 | 东南	48人关注 / 4月前发布	NaN	15569.6
388	锦江	鑫苑名家一期	鑫苑名家一期小套二，户型方正。。。。。。	高楼层(共34层) | 2011年建 | 2室1厅 | 55.72平米 | 东南	4人关注 / 19天前发布	NaN	16152.2
458	锦江	鑫苑名家一期	装修保养好，配套成熟，交通出行方便，	低楼层(共35层) | 2011年建 | 1室1厅 | 41平米 | 南	10人关注 / 4月前发布	NaN	6585.4
463	锦江	鑫苑名家一期	鑫苑名家一期 1室0厅 东南	高楼层(共35层) | 2011年建 | 1室0厅 | 43平米 | 东南	5人关注 / 6月前发布	NaN	6976.7
474	锦江	鑫苑名家一期	鑫苑名家一期，新青年高楼层公寓出售，带家具家电	高楼层(共35层) | 2011年建 | 1室1厅 | 41平米 | 东北	15人关注 / 1年前发布	NaN	6585.4
476	锦江	鑫苑名家一期	房子楼层低，出行方便，主卧带阳台，空间利用合理	低楼层(共34层) | 2011年建 | 2室1厅 | 63平米 | 东	14人关注 / 8月前发布	NaN	16190.5
551	锦江	鑫苑名家一期	中间楼层，正看中庭视野好，采光无遮挡，合理利用空间	高楼层(共34层) | 2011年建 | 2室2厅 | 55.52平米 | 东南	3人关注 / 6月前发布	NaN	16570.6

562	锦江	鑫苑名家一期	鑫苑名家一期 大套三 户型通透	中楼层(共34层) | 2011年建 | 3室1厅 | 104平米 | 东南	11人关注 / 11月前发布	NaN	19230.8
576	锦江	鑫苑名家一期	鑫苑名家一期，带装修套二，中间楼层	中楼层(共34层) | 2011年建 | 2室1厅 | 61.07平米 | 东南	31人关注 / 3月前发布	NaN	15555.9
638	锦江	鑫苑名家一期	鑫苑名家一期标准套二，居家装修对中庭	高楼层(共34层) | 2011年建 | 2室1厅 | 59.45平米 | 东南	75人关注 / 4月前发布	NaN	15307.0
651	锦江	鑫苑名家一期	鑫苑名家 精装套二变换套三 朝南对中庭	高楼层(共34层) | 2011年建 | 3室2厅 | 79.64平米 | 东南	26人关注 / 9月前发布	NaN	17202.4
711	锦江	鑫苑名家一期	鑫苑名家一期套三，，，，，，，，	中楼层(共34层) | 2011年建 | 3室1厅 | 79平米 | 西南	7人关注 / 12月前发布	NaN	17721.5
722	锦江	鑫苑名家一期	此房业主诚心出售 楼层不高 出行方便 可做经营项目	低楼层(共34层) | 2011年建 | 2室1厅 | 69平米 | 东南	6人关注 / 7月前发布	NaN	15942.0
813	锦江	鑫苑名家一期	鑫苑名家一期，居家装修，小套二，中底楼层，采光好	低楼层(共34层) | 2011年建 | 2室2厅 | 62.7平米 | 东南	3人关注 / 2月前发布	NaN	16746.4
870	锦江	鑫苑名家一期	永辉超市旁 自住精装套二 中间楼层 对中庭	高楼层(共34层) | 2011年建 | 2室1厅 | 62.46平米 | 南	12人关注 / 7月前发布	NaN	16810.8
899	锦江	鑫苑名家一期	鑫苑名家、精装套三、中间楼层、看房方便	中楼层(共34层) | 2011年建 | 3室1厅 | 79平米 | 南	5人关注 / 4月前发布	NaN	18987.3
1017	锦江	鑫苑名家一期	鑫苑名家一期新青年，中高楼层精装修公寓	中楼层(共35层) | 2011年建 | 1室1厅 | 40.65平米 | 南	11人关注 / 11月前发布	NaN	6642.1
1026	锦江	鑫苑名家一期	有装修 中间楼层标间 采光好 位置安静	高楼层(共35层) | 2011年建 | 1室0厅 | 41平米 | 东南	0人关注 / 2月前发布	NaN	7073.2
1029	锦江	鑫苑名家一期	锦江区 三圣乡 鑫苑名家一期 自住装修带车位 诚心出售	中楼层(共34层) | 2011年建 | 3室1厅 | 79.66平米 | 东南	11人关注 / 7月前发布	NaN	20085.4

【应用实例】对二手房数据进行清洗

```
# （7）用箱线图查看以上该小区的单价异常值
from matplotlib import pyplot as plt


plt.rcParams['font.sans-serif'] = ['SimHei']
plt.boxplot(xq['单价(元/平米)'])
plt.show()
```



【应用实例】对二手房数据进行清洗

```
# （8）查看‘鑫苑名家一期’的统计信息
xq = data[data['小区名称'] == '鑫苑名家一期']
dj = xq['单价(元/平米)']
c = dj.describe()
print(c)
```

```
count       20.000000
mean     14452.530000
std       4714.064684
min       6585.400000
25%      13248.550000
50%      16171.350000
75%      17137.150000
max      20085.400000
IQR       3888.600000
Name: 单价(元/平米), dtype: float64
```

【应用实例】对二手房数据进行清洗

```
#  （9）删除'鑫苑名家一期'的异常值
c.loc['IQR']= c.loc['75%']-c.loc['25%']  # 四分位距IQR
low = c.loc['25%'] - 1.5*c.loc['IQR']    # 正常值下界
upper = c.loc['75%'] + 1.5*c.loc['IQR'] # 正常值上界
yc = data[(data['小区名称'] == '鑫苑名家一期') & \
          ((data['单价(元/平米)']>upper) | (data['单价(元/平米)']<low))]
data = data.drop(yc.index)
print(data[data['小区名称']=='鑫苑名家一期'])
```

	区	小区名称	标题	房屋信息	关注	地铁	单价(元/平米)
150	锦江	鑫苑名家一期	鑫苑名家一期 3室1厅 东南	高楼层(共34层) | 2011年建 | 3室1厅 | 104平米 | 东南	22人关注 / 2月前发布	NaN	17115.4
213	锦江	鑫苑名家一期	鑫苑名家一期，套二带装修，中间楼层，	高楼层(共34层) | 2011年建 | 2室1厅 | 79平米 | 东南	48人关注 / 4月前发布	NaN	15569.6
388	锦江	鑫苑名家一期	鑫苑名家一期小套二，户型方正。。。。。。	高楼层(共34层) | 2011年建 | 2室1厅 | 55.72平米 | 东南	4人关注 / 19天前发布	NaN	16152.2
476	锦江	鑫苑名家一期	房子楼层低，出行方便，主卧带阳台，空间利用合理	低楼层(共34层) | 2011年建 | 2室1厅 | 63平米 | 东	14人关注 / 8月前发布	NaN	16190.5
551	锦江	鑫苑名家一期	中间楼层，正看中庭视野好，采光无遮挡，合理利用空间	高楼层(共34层) | 2011年建 | 2室2厅 | 55.52平米 | 东南	3人关注 / 6月前发布	NaN	16570.6
562	锦江	鑫苑名家一期	鑫苑名家一期 大套三 户型通透	中楼层(共34层) | 2011年建 | 3室1厅 | 104平米 | 东南	11人关注 / 11月前发布	NaN	19230.8
576	锦江	鑫苑名家一期	鑫苑名家一期，带装修套二，中间楼层	中楼层(共34层) | 2011年建 | 2室1厅 | 61.07平米 | 东南	31人关注 / 3月前发布	NaN	15555.9
638	锦江	鑫苑名家一期	鑫苑名家一期标准套二，居家装修对中庭	高楼层(共34层) | 2011年建 | 2室1厅 | 59.45平米 | 东南	75人关注 / 4月前发布	NaN	15307.0
651	锦江	鑫苑名家一期	鑫苑名家 精装套二变换套三 朝南对中庭	高楼层(共34层) | 2011年建 | 3室2厅 | 79.64平米 | 东南	26人关注 / 9月前发布	NaN	17202.4
711	锦江	鑫苑名家一期	鑫苑名家一期套三，，，，，，，	中楼层(共34层) | 2011年建 | 3室1厅 | 79平米 | 西南	7人关注 / 12月前发布	NaN	17721.5
722	锦江	鑫苑名家一期	此房业主诚心出售 楼层不高 出行方便 可做经营项目	低楼层(共34层) | 2011年建 | 2室1厅 | 69平米 | 东南	6人关注 / 7月前发布	NaN	15942.0
813	锦江	鑫苑名家一期	鑫苑名家一期，居家装修，小套二，中底楼层，采光好	低楼层(共34层) | 2011年建 | 2室2厅 | 62.7平米 | 东南	3人关注 / 2月前发布	NaN	16746.4
870	锦江	鑫苑名家一期	永辉超市旁 自住精装套二 中间楼层 对中庭	高楼层(共34层) | 2011年建 | 2室1厅 | 62.46平米 | 南	12人关注 / 7月前发布	NaN	16810.8
899	锦江	鑫苑名家一期	鑫苑名家、精装套三、中间楼层、看房方便	中楼层(共34层) | 2011年建 | 3室1厅 | 79平米 | 南	5人关注 / 4月前发布	NaN	18987.3
1029	锦江	鑫苑名家一期	锦江区 三圣乡 鑫苑名家一期 自住装修带车位 诚心出售	中楼层(共34层) | 2011年建 | 3室1厅 | 79.66平米 | 东南	11人关注 / 7月前发布	NaN	20085.4

END