数据预处理

内容

- 数据导入
- 数据观察
- 数据预处理
- 数据导出

数据导入：导入.csv文件

pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None,
header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=False,
prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None, skipinitialspace=False,
skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True,
na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False,
infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None,
decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True,
escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None, storage_options=None)      [source]
```

Kaggle： BeijingPM20100101_20151231.csv

No	year	month	day	hour	season	PM_Dongs	PM_Dongs	PM_Nongz	PM_US Pos	DEWP	HUMI	PRES	TEMP	cbwd	Iws	precipitatio	Iprec
1	2010	1	1	0	4	NA	NA	NA	NA	-21	43	1021	-11	NW	1.79	0	0
2	2010	1	1	1	4	NA	NA	NA	NA	-21	47	1020	-12	NW	4.92	0	0
3	2010	1	1	2	4	NA	NA	NA	NA	-21	43	1019	-11	NW	6.71	0	0
4	2010	1	1	3	4	NA	NA	NA	NA	-21	55	1019	-14	NW	9.84	0	0
5	2010	1	1	4	4	NA	NA	NA	NA	-20	51	1018	-12	NW	12.97	0	0
6	2010	1	1	5	4	NA	NA	NA	NA	-19	47	1017	-10	NW	16.1	0	0
7	2010	1	1	6	4	NA	NA	NA	NA	-19	44	1017	-9	NW	19.23	0	0
8	2010	1	1	7	4	NA	NA	NA	NA	-19	44	1017	-9	NW	21.02	0	0
9	2010	1	1	8	4	NA	NA	NA	NA	-19	44	1017	-9	NW	24.15	0	0
10	2010	1	1	9	4	NA	NA	NA	NA	-20	37	1017	-8	NW	27.28	0	0
11	2010	1	1	10	4	NA	NA	NA	NA	-19	37	1017	-7	NW	31.3	0	0
12	2010	1	1	11	4	NA	NA	NA	NA	-18	35	1017	-5	NW	34.43	0	0
13	2010	1	1	12	4	NA	NA	NA	NA	-19	32	1015	-5	NW	37.56	0	0
14	2010	1	1	13	4	NA	NA	NA	NA	-18	30	1015	-3	NW	40.69	0	0
15	2010	1	1	14	4	NA	NA	NA	NA	-18	28	1014	-2	NW	43.82	0	0
16	2010	1	1	15	4	NA	NA	NA	NA	-18	26	1014	-1	cv	0.89	0	0
17	2010	1	1	16	4	NA	NA	NA	NA	-19	25	1015	-2	NW	1.79	0	0
18	2010	1	1	17	4	NA	NA	NA	NA	-18	30	1015	-3	NW	2.68	0	0
19	2010	1	1	18	4	NA	NA	NA	NA	-18	35	1016	-5	NE	1.79	0	0
20	2010	1	1	19	4	NA	NA	NA	NA	-17	35	1017	-4	NW	1.79	0	0
21	2010	1	1	20	4	NA	NA	NA	NA	-17	38	1017	-5	cv	0.89	0	0
22	2010	1	1	21	4	NA	NA	NA	NA	-17	38	1018	-5	NW	1.79	0	0
23	2010	1	1	22	4	NA	NA	NA	NA	-17	38	1018	-5	NW	2.68	0	0
24	2010	1	1	23	4	NA	NA	NA	129	-17	41	1020	-5	cv	0.89	0	0
25	2010	1	2	0	4	NA	NA	NA	148	-16	38	1020	-4	SE	1.79	0	0
26	2010	1	2	1	4	NA	NA	NA	159	-15	42	1020	-4	SE	2.68	0	0
27	2010	1	2	2	4	NA	NA	NA	181	-11	63.5	1021	-5	SE	3.57	0	0
28	2010	1	2	3	4	NA	NA	NA	138	-7	85	1022	-5	SE	5.36	0	0
29	2010	1	2	4	4	NA	NA	NA	109	-7	85	1022	-5	SE	6.25	0	0
30	2010	1	2	5	4	NA	NA	NA	105	-7	92	1022	-6	SE	7.14	0	0
31	2010	1	2	6	4	NA	NA	NA	124	-7	92	1023	-6	SE	8.93	0	0
32	2010	1	2	7	4	NA	NA	NA	120	-7	85	1024	-5	SE	10.72	0	0
33	2010	1	2	8	4	NA	NA	NA	132	-8	85	1024	-6	SE	12.51	0	0
34	2010	1	2	9	4	NA	NA	NA	140	-7	85	1025	-5	SE	14.3	0	0
35	2010	1	2	10	4	NA	NA	NA	152	-7	85	1026	-5	SE	17.43	0	0
36	2010	1	2	11	4	NA	NA	NA	148	-8	79	1026	-5	SE	20.56	0	0

数据导入：导入.csv文件

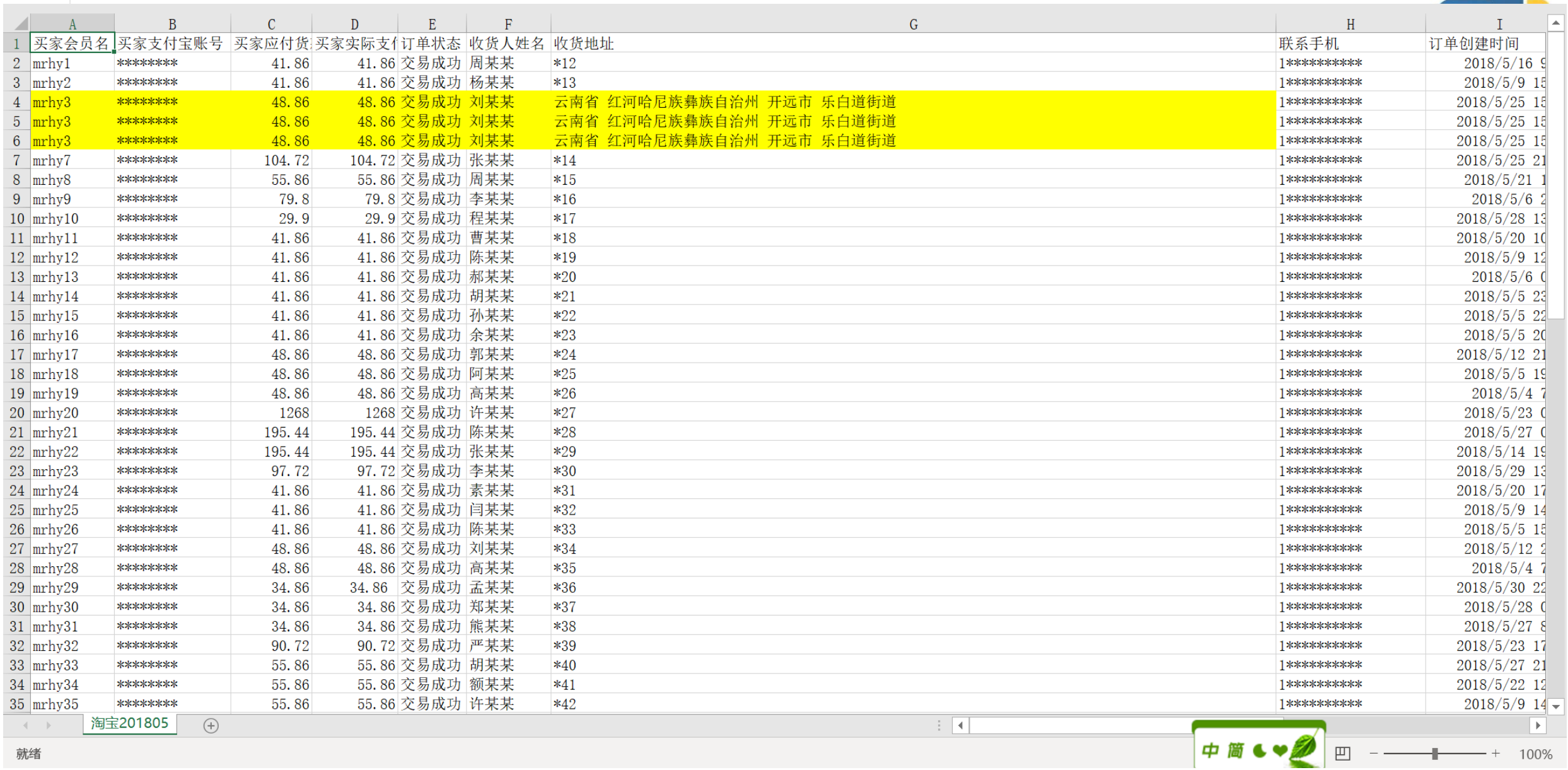
```
import pandas as pd
# 设置数据显示的最大列数和宽度
pd.set_option('display.max_columns',500)
pd.set_option('display.width',1000)
# 解决输出时列名不对齐问题
pd.set_option('display.unicode.east_asian_width',True)
pm_beijing = pd.read_csv('BeijingPM20100101_20151231.csv')
```

```
>>> pm_beijing
          No  year  month  day  hour  season  PM_Dongsi  PM_Dongsihuan  PM_Nongzhanguan  PM_US Post  DEWP  HUMI    PRES  TEMP cbwd    Iws  precipitation  Iprec
0          1  2010      1    1     0       4        NaN            NaN              NaN         NaN -21.0  43.0  1021.0 -11.0   NW   1.79            0.0    0.0
1          2  2010      1    1     1       4        NaN            NaN              NaN         NaN -21.0  47.0  1020.0 -12.0   NW   4.92            0.0    0.0
2          3  2010      1    1     2       4        NaN            NaN              NaN         NaN -21.0  43.0  1019.0 -11.0   NW   6.71            0.0    0.0
3          4  2010      1    1     3       4        NaN            NaN              NaN         NaN -21.0  55.0  1019.0 -14.0   NW   9.84            0.0    0.0
4          5  2010      1    1     4       4        NaN            NaN              NaN         NaN -20.0  51.0  1018.0 -12.0   NW  12.97            0.0    0.0
      ...   ...    ...  ...   ...     ...        ...            ...              ...         ...   ...   ...     ...   ...  ...   ...            ...    ...
52579  52580  2015     12   31    19       4      140.0          157.0            122.0       133.0  -8.0  68.0  1031.0  -3.0   SE   7.14            0.0    0.0
52580  52581  2015     12   31    20       4      157.0          199.0            149.0       169.0  -8.0  63.0  1030.0  -2.0   SE   8.03            0.0    0.0
```

数据导入：导入.xls、xlsx文件

pandas.read_excel

```
pandas.read_excel(io, sheet_name=0, header=0, names=None, index_col=None, usecols=None,
squeeze=False, dtype=None, engine=None, converters=None, true_values=None,
false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True,
na_filter=True, verbose=False, parse_dates=False, date_parser=None, thousands=None,
comment=None, skipfooter=0, convert_float=None, mangle_dupe_cols=True,
storage_options=None)                                                    [source]
```



	A	B	C	D	E	F	G	H	I
1	买家会员名	买家支付宝账号	买家应付货	买家实际支付	订单状态	收货人姓名	收货地址	联系手机	订单创建时间
2	mrhy1	********	41.86	41.86	交易成功	周某某	*12	1**********	2018/5/16 9
3	mrhy2	********	41.86	41.86	交易成功	杨某某	*13	1**********	2018/5/9 15
4	mrhy3	********	48.86	48.86	交易成功	刘某某	云南省 红河哈尼族彝族自治州 开远市 乐白道街道	1**********	2018/5/25 15
5	mrhy3	********	48.86	48.86	交易成功	刘某某	云南省 红河哈尼族彝族自治州 开远市 乐白道街道	1**********	2018/5/25 15
6	mrhy3	********	48.86	48.86	交易成功	刘某某	云南省 红河哈尼族彝族自治州 开远市 乐白道街道	1**********	2018/5/25 15
7	mrhy7	********	104.72	104.72	交易成功	张某某	*14	1**********	2018/5/25 21
8	mrhy8	********	55.86	55.86	交易成功	周某某	*15	1**********	2018/5/21 1
9	mrhy9	********	79.8	79.8	交易成功	李某某	*16	1**********	2018/5/6 2
10	mrhy10	********	29.9	29.9	交易成功	程某某	*17	1**********	2018/5/28 13
11	mrhy11	********	41.86	41.86	交易成功	曹某某	*18	1**********	2018/5/20 10
12	mrhy12	********	41.86	41.86	交易成功	陈某某	*19	1**********	2018/5/9 12
13	mrhy13	********	41.86	41.86	交易成功	郝某某	*20	1**********	2018/5/6 0
14	mrhy14	********	41.86	41.86	交易成功	胡某某	*21	1**********	2018/5/5 23
15	mrhy15	********	41.86	41.86	交易成功	孙某某	*22	1**********	2018/5/5 22
16	mrhy16	********	41.86	41.86	交易成功	余某某	*23	1**********	2018/5/5 20
17	mrhy17	********	48.86	48.86	交易成功	郭某某	*24	1**********	2018/5/12 21
18	mrhy18	********	48.86	48.86	交易成功	阿某某	*25	1**********	2018/5/5 19
19	mrhy19	********	48.86	48.86	交易成功	高某某	*26	1**********	2018/5/4 7
20	mrhy20	********	1268	1268	交易成功	许某某	*27	1**********	2018/5/23 0
21	mrhy21	********	195.44	195.44	交易成功	陈某某	*28	1**********	2018/5/27 0
22	mrhy22	********	195.44	195.44	交易成功	张某某	*29	1**********	2018/5/14 19
23	mrhy23	********	97.72	97.72	交易成功	李某某	*30	1**********	2018/5/29 13
24	mrhy24	********	41.86	41.86	交易成功	素某某	*31	1**********	2018/5/20 17
25	mrhy25	********	41.86	41.86	交易成功	闫某某	*32	1**********	2018/5/9 14
26	mrhy26	********	41.86	41.86	交易成功	陈某某	*33	1**********	2018/5/5 15
27	mrhy27	********	48.86	48.86	交易成功	刘某某	*34	1**********	2018/5/12 2
28	mrhy28	********	48.86	48.86	交易成功	高某某	*35	1**********	2018/5/4 7
29	mrhy29	********	34.86	34.86	交易成功	孟某某	*36	1**********	2018/5/30 22
30	mrhy30	********	34.86	34.86	交易成功	郑某某	*37	1**********	2018/5/28 0
31	mrhy31	********	34.86	34.86	交易成功	熊某某	*38	1**********	2018/5/27 8
32	mrhy32	********	90.72	90.72	交易成功	严某某	*39	1**********	2018/5/23 17
33	mrhy33	********	55.86	55.86	交易成功	胡某某	*40	1**********	2018/5/27 21
34	mrhy34	********	55.86	55.86	交易成功	额某某	*41	1**********	2018/5/22 12
35	mrhy35	********	55.86	55.86	交易成功	许某某	*42	1**********	2018/5/9 14

数据导入：导入.xls、xlsx文件

```
>>> import openpyxl
>>> df5 = pd.read_excel('1月.xlsx')
>>> print(df5)
    买家会员名    买家支付宝账号  买家应付货款  买家实际支付金额  ...          订单备注  宝贝总数量     类别  图书编号
0     mrhy1  ********   41.86     41.86  ...          'null     1  全彩系列  B16
1     mrhy2  ********   41.86     41.86  ...          'null     1  全彩系列  B16
2     mrhy3  ********   48.86     48.86  ...          'null     1  全彩系列  B17
3     mrhy3  ********   48.86     48.86  ...          'null     1  全彩系列  B17
4     mrhy3  ********   48.86     48.86  ...          'null     1  全彩系列  B17
..      ...       ...     ...       ...  ...            ...   ...   ...  ...
75   mrhy77  ********  268.00    268.00  ...  '中通：493598151449     1   NaN  NaN
76   mrhy78  ********  268.00    268.00  ...          'null     1   NaN  NaN
77   mrhy79  ********  268.00    268.00  ...  '中通：492021024200     1   NaN  NaN
78   mrhy80  ********  268.00    268.00  ...  '中通：493781830401     1   NaN  NaN
79   mrhy81  ********  268.00    268.00  ...  '中通：632394471077     1   NaN  NaN

[80 rows x 16 columns]
```

内容

- 数据导入
- 数据观察
- 数据预处理
- 数据导出

数据观察

df.head(n)

```
>>> print(pm_beijing.head(10))
   No  year  month  day  hour  season  PM_Dongsi  PM_Dongsihuan  PM_Nongzhanguan  PM_US Post  DEWP  HUMI    PRES  TEMP cbwd    Iws  precipitation  Iprec
0   1  2010      1    1     0       4        NaN            NaN              NaN         NaN -21.0  43.0  1021.0 -11.0   NW   1.79            0.0    0.0
1   2  2010      1    1     1       4        NaN            NaN              NaN         NaN -21.0  47.0  1020.0 -12.0   NW   4.92            0.0    0.0
2   3  2010      1    1     2       4        NaN            NaN              NaN         NaN -21.0  43.0  1019.0 -11.0   NW   6.71            0.0    0.0
3   4  2010      1    1     3       4        NaN            NaN              NaN         NaN -21.0  55.0  1019.0 -14.0   NW   9.84            0.0    0.0
4   5  2010      1    1     4       4        NaN            NaN              NaN         NaN -20.0  51.0  1018.0 -12.0   NW  12.97            0.0    0.0
5   6  2010      1    1     5       4        NaN            NaN              NaN         NaN -19.0  47.0  1017.0 -10.0   NW  16.10            0.0    0.0
6   7  2010      1    1     6       4        NaN            NaN              NaN         NaN -19.0  44.0  1017.0  -9.0   NW  19.23            0.0    0.0
7   8  2010      1    1     7       4        NaN            NaN              NaN         NaN -19.0  44.0  1017.0  -9.0   NW  21.02            0.0    0.0
8   9  2010      1    1     8       4        NaN            NaN              NaN         NaN -19.0  44.0  1017.0  -9.0   NW  24.15            0.0    0.0
9  10  2010      1    1     9       4        NaN            NaN              NaN         NaN -20.0  37.0  1017.0  -8.0   NW  27.28            0.0    0.0
```

数据观察

df.tail(n)

```
>>> print(pm_beijing.tail())
          No  year  month  day  hour  season  PM_Dongsi  PM_Dongsihuan  PM_Nongzhanguan  PM_US Post  DEWP  HUMI    PRES  TEMP cbwd   Iws  precipitation  Iprec
52579  52580  2015     12   31    19       4      140.0          157.0            122.0       133.0  -8.0  68.0  1031.0  -3.0   SE  7.14            0.0    0.0
52580  52581  2015     12   31    20       4      157.0          199.0            149.0       169.0  -8.0  63.0  1030.0  -2.0   SE  8.03            0.0    0.0
52581  52582  2015     12   31    21       4      171.0          231.0            196.0       203.0 -10.0  73.0  1030.0  -6.0   NE  0.89            0.0    0.0
52582  52583  2015     12   31    22       4      204.0          242.0            221.0       212.0 -10.0  73.0  1030.0  -6.0   NE  1.78            0.0    0.0
52583  52584  2015     12   31    23       4        NaN            NaN              NaN       235.0  -9.0  79.0  1029.0  -6.0   NE  2.67            0.0    0.0
```

数据观察

df.sample(n)

```
>>> print(pm_beijing.sample(8))
```

	No	year	month	day	hour	season	PM_Dongsi	PM_Dongsihuan	PM_Nongzhanguan	PM_US Post	DEWP	HUMI	PRES	TEMP	cbwd	Iws	precipitation	Iprec
24197	24198	2012	10	5	5	3	NaN	NaN	NaN	17.0	9.0	87.0	1016.0	11.0	NW	0.89	0.0	0.0
37978	37979	2014	5	2	10	1	22.0	16.0	21.0	50.0	-7.0	16.0	1018.0	19.0	NW	171.65	0.0	0.0
19267	19268	2012	3	13	19	1	NaN	NaN	NaN	81.0	-14.0	13.0	1016.0	13.0	SE	39.78	0.0	0.0
26548	26549	2013	1	11	4	4	NaN	NaN	NaN	429.0	-7.0	85.0	1024.0	-5.0	NW	2.68	0.0	0.0
13112	13113	2011	7	1	8	2	NaN	NaN	NaN	82.0	23.0	88.0	1004.0	25.0	SE	4.92	3.5	6.0
50672	50673	2015	10	13	8	3	28.0	NaN	28.0	30.0	5.0	54.0	1022.0	14.0	NW	19.68	0.0	0.0
8036	8037	2010	12	1	20	4	NaN	NaN	NaN	288.0	-1.0	86.0	1012.0	1.0	NW	1.79	0.0	0.0
7128	7129	2010	10	25	0	3	NaN	NaN	NaN	10.0	-3.0	52.0	1031.0	6.0	NW	22.35	0.0	0.0

数据观察

df.info()

```
>>> pm_beijing.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52584 entries, 0 to 52583
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   No               52584 non-null  int64
 1   year             52584 non-null  int64
 2   month            52584 non-null  int64
 3   day              52584 non-null  int64
 4   hour             52584 non-null  int64
 5   season           52584 non-null  int64
 6   PM_Dongsi        25052 non-null  float64
 7   PM_Dongsihuan    20508 non-null  float64
 8   PM_Nongzhanguan  24931 non-null  float64
 9   PM_US Post       50387 non-null  float64
 10  DEWP             52579 non-null  float64
 11  HUMI             52245 non-null  float64
 12  PRES             52245 non-null  float64
 13  TEMP             52579 non-null  float64
 14  cbwd             52579 non-null  object
 15  Iws              52579 non-null  float64
 16  precipitation    52100 non-null  float64
 17  Iprec            52100 non-null  float64
dtypes: float64(11), int64(6), object(1)
memory usage: 7.2+ MB
```


```
RangeIndex: 52584 entries, 0 to 52583
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   No               52584 non-null  int64
 1   year             52584 non-null  int64
 2   month            52584 non-null  int64
 3   day              52584 non-null  int64
 4   hour             52584 non-null  int64
 5   season           52584 non-null  int64
 6   PM_Dongsi        25052 non-null  float64
 7   PM_Dongsihuan    20508 non-null  float64
 8   PM_Nongzhanguan  24931 non-null  float64
 9   PM_US Post       50387 non-null  float64
 10  DEWP             52579 non-null  float64
 11  HUMI             52245 non-null  float64
 12  PRES             52245 non-null  float64
 13  TEMP             52579 non-null  float64
 14  cbwd             52579 non-null  object
 15  Iws              52579 non-null  float64
 16  precipitation    52100 non-null  float64
 17  Iprec            52100 non-null  float64
```

数据观察

df.describe()

```
>>> pm_beijing.describe()
                 No          year
count  52584.000000  52584.000000  52584.
mean   26292.500000   2012.499772      6.
std    15179.837614      1.707485      3.
min        1.000000   2010.000000      1.
25%    13146.750000   2011.000000      4.
50%    26292.500000   2012.000000      7.
75%    39438.250000   2014.000000     10.
max    52584.000000   2015.000000     12.
```

```
>>> pm_beijing.describe()
                 No          year         month           day          hour       season     PM_Dongsi  PM_Dongsihuan  PM_Nongzhanguan     PM_US Post           DEWP          H
count  52584.000000  52584.000000  52584.000000  52584.000000  52584.000000  52584.000000  25052.000000   20508.000000     24931.000000   50387.000000   52579.000000   52245.000
mean   26292.500000   2012.499772      6.523962     15.726609     11.500000      2.491100     89.154439      92.560806        88.643737      95.904241       2.074554      54.602
std    15179.837614      1.707485      3.448452      8.798896      6.922252      1.116988     87.239267      88.027434        88.041166      91.643772      14.222059      25.991
min        1.000000   2010.000000      1.000000      1.000000      0.000000      1.000000      3.000000       3.000000         3.000000       1.000000     -40.000000       2.000
25%    13146.750000   2011.000000      4.000000      8.000000      5.750000      1.000000     24.000000      28.000000        24.000000      27.000000     -10.000000      31.000
50%    26292.500000   2012.000000      7.000000     16.000000     11.500000      2.000000     64.000000      68.000000        62.000000      69.000000       2.000000      55.000
75%    39438.250000   2014.000000     10.000000     23.000000     17.250000      3.000000    124.000000     127.000000       122.000000     132.000000      15.000000      78.000
max    52584.000000   2015.000000     12.000000     31.000000     23.000000      4.000000    737.000000     672.000000       844.000000     994.000000      28.000000     100.000
```

数据观察

```
>>> pm_beijing.iloc[:,6:17].describe()
```

	PM_Dongsi	PM_Dongsihuan	PM_Nongzhanguan	PM_US Post	DEWP	HUMI	PRES	TEMP	Iws	precipitation
count	25052.000000	20508.000000	24931.000000	50387.000000	52579.000000	52245.000000	52245.000000	52579.000000	52579.000000	52100.000000
mean	89.154439	92.560806	88.643737	95.904241	2.074554	54.602421	1016.465442	12.587040	23.261829	19.258683
std	87.239267	88.027434	88.041166	91.643772	14.222059	25.991338	10.295070	12.098527	49.281706	4381.035532
min	3.000000	3.000000	3.000000	1.000000	-40.000000	2.000000	991.000000	-19.000000	0.450000	0.000000
25%	24.000000	28.000000	24.000000	27.000000	-10.000000	31.000000	1008.000000	2.000000	1.790000	0.000000
50%	64.000000	68.000000	62.000000	69.000000	2.000000	55.000000	1016.000000	14.000000	4.920000	0.000000
75%	124.000000	127.000000	122.000000	132.000000	15.000000	78.000000	1025.000000	23.000000	21.020000	0.000000
max	737.000000	672.000000	844.000000	994.000000	28.000000	100.000000	1046.000000	42.000000	585.600000	999990.000000


```
>>> pm_beijing.iloc[:,6:17].describe()
```


	PM_Dongsi	PM_Dongsihuan	PM_Nongzhanguan	PM_US Post	DEWP	HUMI
count	25052.000000	20508.000000	24931.000000	50387.000000	52579.000000	52245.000000
mean	89.154439	92.560806	88.643737	95.904241	2.074554	54.602421
std	87.239267	88.027434	88.041166	91.643772	14.222059	25.991338
min	3.000000	3.000000	3.000000	1.000000	-40.000000	2.000000
25%	24.000000	28.000000	24.000000	27.000000	-10.000000	31.000000
50%	64.000000	68.000000	62.000000	69.000000	2.000000	55.000000
75%	124.000000	127.000000	122.000000	132.000000	15.000000	78.000000
max	737.000000	672.000000	844.000000	994.000000	28.000000	100.000000

数据观察

```
>>> pm_beijing.loc[pm_beijing['year'] == 2015].iloc[:,6:17].describe()
         PM_Dongsi  PM_Dongsihuan  PM_Nongzhanguan   PM_US Post          DEWP          HUMI
count  8596.000000    5465.000000      8473.000000  8631.000000  8755.000000  8421.000000
mean     87.492206      89.359927        86.100083    82.784729     3.362536    56.146538
std      92.504211      91.852083        91.100675    88.492918    13.037508    26.014223
min       3.000000       3.000000         3.000000     1.000000   -30.000000     5.000000
25%      22.000000      25.000000        21.000000    22.000000    -8.000000    34.000000
50%      58.000000      64.000000        56.000000    55.000000     4.000000    57.000000
75%     118.000000     120.000000       117.000000   109.000000    15.000000    79.000000
max     685.000000     671.000000       667.000000   722.000000    27.000000   100.000000
```

内容

- 数据导入
- 数据观察
- 数据预处理
- 数据导出

数据预处理的概念

- 在数据分析和数据挖掘中，初始数据一般来自多数据源且格式多样化，数据质量也良莠不齐，不能直接被应用到数据分析、数据挖掘或机器学习中
- 数据预处理是数据分析或数据挖掘前的准备工作，也是数据分析或数据挖掘中必不可少的一环
- 数据预处理通过一系列的方法处理“脏”数据、精准地抽取数据、调整数据格式，从而得到一组符合准确、完整、简洁等标准的高质量数据，以更好地服务于数据分析或数据挖掘工作

常见的数据问题

- 数据缺失：部分数据值为空
- 数据重复：同一条数据多次出现
- 数据异常：个别数据远离数据集
- 数据冗余：数据中存在一些多余、无意义的属性
- 数据值冲突：同一属性存在不同值
- 数据噪声：属性值不符合常理

数据预处理方法

- 数据清理
- 数据变换
- 数据规约
- 数据集成
- 数据格式转换

数据清理

数据清理将“脏”数据清理成质量较高的“干净”数据

- 缺失值的检测与处理
- 重复值的检测与处理
- 异常值的检测与处理

数据清理：缺失值I

df.info()

```
>>> df3 = pd.read_csv('lianjia.csv',encoding ='utf-8')
>>> df3.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23677 entries, 0 to 23676
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Direction    23677 non-null  object
 1   District     23677 non-null  object
 2   Elevator     15440 non-null  object
 3   Floor        23677 non-null  int64
 4   Garden       23677 non-null  object
 5   Id           23677 non-null  int64
 6   Layout       23677 non-null  object
 7   Price        23677 non-null  float64
 8   Region       23677 non-null  object
 9   Renovation   23677 non-null  object
 10  Size         23677 non-null  float64
 11  Year         23677 non-null  int64
dtypes: float64(2), int64(3), object(7)
```

数据清理：缺失值的检测

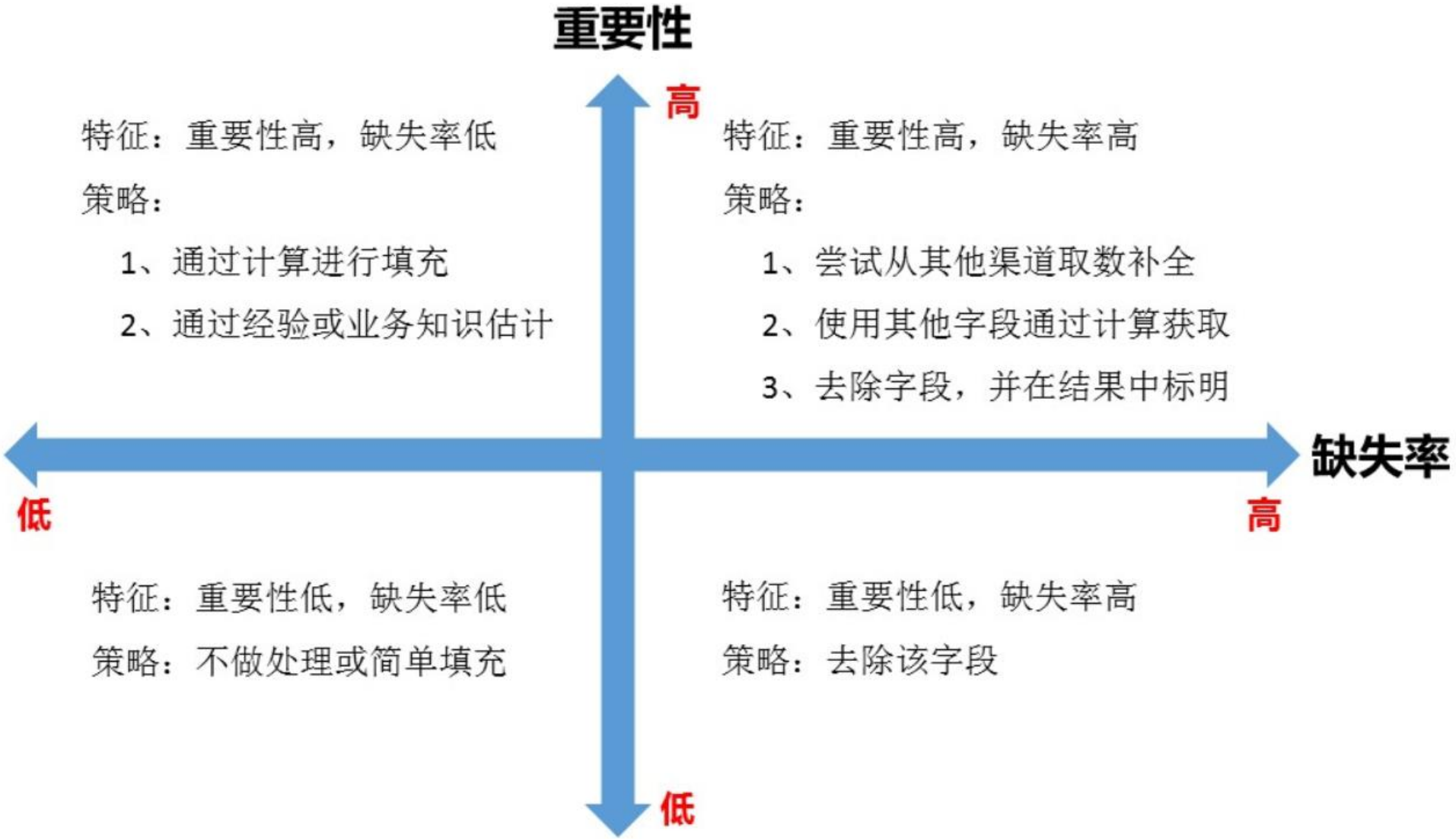
df.isnull(), df.isna()

df.notull(), df.notna()

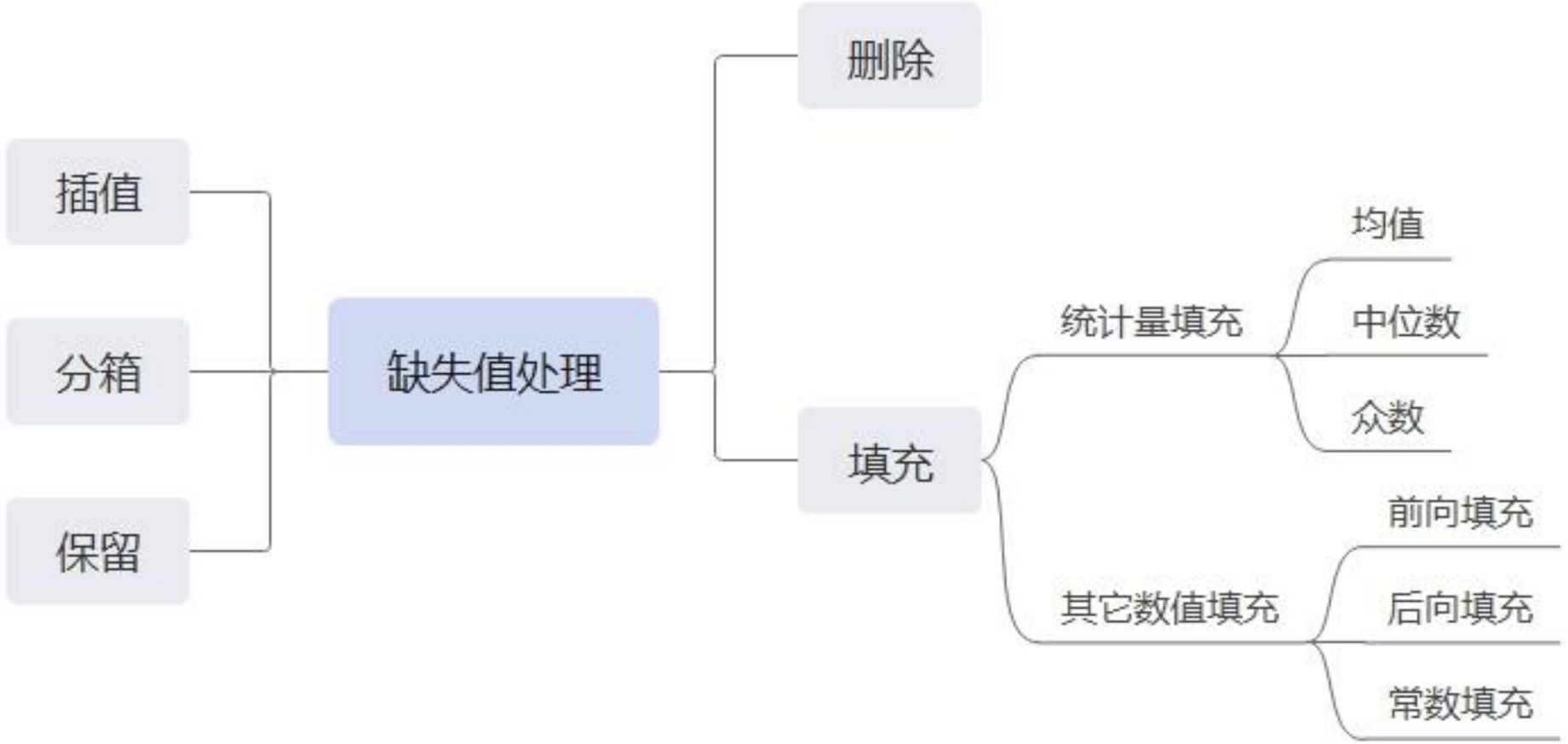
```
>>> dict4 = {'A':[1,2,np.NaN,4],'B':[3,4,4,5],'C':[5,6,7,8],'D':[7,5,np.NaN,np.NaN]}
>>> df4 = pd.DataFrame(dict4)
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> df4.isna()
       A      B      C      D
0  False  False  False  False
1  False  False  False  False
2   True  False  False   True
3  False  False  False   True
```

数据清理：缺失值的处理


重要性
高
低
缺失率
低
高
特征：重要性高，缺失率低
策略：
1、通过计算进行填充
2、通过经验或业务知识估计
特征：重要性高，缺失率高
策略：
1、尝试从其他渠道取数补全
2、使用其他字段通过计算获取
3、去除字段，并在结果中标明
特征：重要性低，缺失率低
策略：不做处理或简单填充
特征：重要性低，缺失率高
策略：去除该字段

数据清理：缺失值的处理



数据清理：缺失值的处理——删除

df.dropna(axis=0, how= ‘any’ , thresh=None, subset=None, inplace=False)

```
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> df4.dropna()
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
```

```
>>> df4.dropna(thresh=3)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
3  4.0  5  8  NaN
```

数据清理：缺失值的处理——填充

df.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)

```
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> col_a = np.around(np.mean(df4['A']),1)
>>> col_d = np.around(np.mean(df4['D']),1)
>>> df4.fillna({'A':col_a,'D':col_d})
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  2.3  4  7  6.0
3  4.0  5  8  6.0
```

数据清理：缺失值的处理——填充

df.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)

```
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> df4.fillna(method='ffill')
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  2.0  4  7  5.0
3  4.0  5  8  5.0
```

数据清理：缺失值的处理——插补

df.interpolate(method= ‘linear’ , axis=0, limit=None, inplace=False, limit_direction=None, limit_area=None, downcast=None, **kwargs)

```
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> df4.interpolate()
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  3.0  4  7  5.0
3  4.0  5  8  5.0
```

```
>>> df4.interpolate(method='barycentric')
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  3.0  4  7  3.0
3  4.0  5  8  1.0
```

数据清理：重复值的检测

df.duplicated(subset=None, keep= ‘first’)

- subset：识别重复项的列索引或列索引序列，默认使用所有列索引
- keep：如何标识重复项（如出现）
  - ‘first’：第一次出现为False，其后出现的均为True
  - ‘last’：最后一次出现为False，其前出现的均为True
  - False：所有重复项均标识为True

df.duplicated().sum()

数据清理：重复值的检测

```
>>> df = pd.DataFrame({
...     'brand': ['YumYum', 'YumYum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [ 4, 4, 3.5, 15, 5]})
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

[pandas.DataFrame.duplicated — pandas 1.3.3 documentation (pydata.org)]

数据清理：重复值的检测

```
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.duplicated()
0    False
1     True
2    False
3    False
4    False
dtype: bool
```

```
>>> df.duplicated(keep='last')
0     True
1    False
2    False
3    False
4    False
dtype: bool
```

```
>>> df.duplicated(keep=False)
0     True
1     True
2    False
3    False
4    False
dtype: bool
```

df[df.duplicated(keep=False)]

数据清理：重复值的检测

```
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.duplicated(subset='brand')
0    False
1     True
2    False
3     True
4     True
dtype: bool
>>> df.duplicated(subset=['brand','style'])
0    False
1     True
2    False
3    False
4     True
dtype: bool
```

数据清理：重复值的删除

df.drop_duplicates(subset=None, keep= ‘first’ , inplace=False, ignore_index=False)

- subset：识别重复项的列索引或列索引序列，默认识别所有列索引
- keep：如何保留重复项（如出现）
  - ‘first’：仅保留第一次出现的数据项
  - ‘last’：仅保留最后一次出现的数据项
  - False：删除所有重复项
- inplace: 就地删除（True）或返回副本（False）
- ignore_index：删除重复项后是否重建索引为0,1,2...n

数据清理：重复值的删除

```
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.drop_duplicates()
     brand style  rating
0   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.drop_duplicates(ignore_index=True)
     brand style  rating
0   YumYum   cup     4.0
1  Indomie   cup     3.5
2  Indomie  pack    15.0
3  Indomie  pack     5.0
```

数据清理：重复值的删除

```
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.drop_duplicates(subset='brand')
     brand style  rating
0   YumYum   cup     4.0
2  Indomie   cup     3.5
```

```
>>> df.drop_duplicates(subset=['brand','style'],keep='last')
     brand style  rating
1   YumYum   cup     4.0
2  Indomie   cup     3.5
4  Indomie  pack     5.0
```

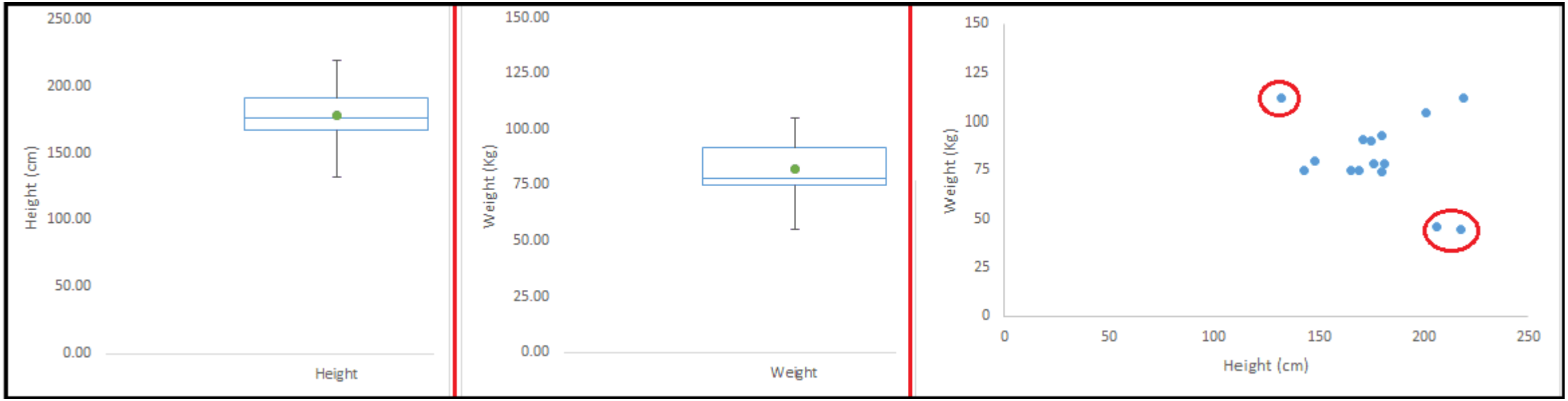
数据清理：重复值的删除

```
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
1   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
>>> df.drop_duplicates(inplace=True)
>>> print(df)
     brand style  rating
0   YumYum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

数据清理：异常值检测

异常值指不在正常范围内的值

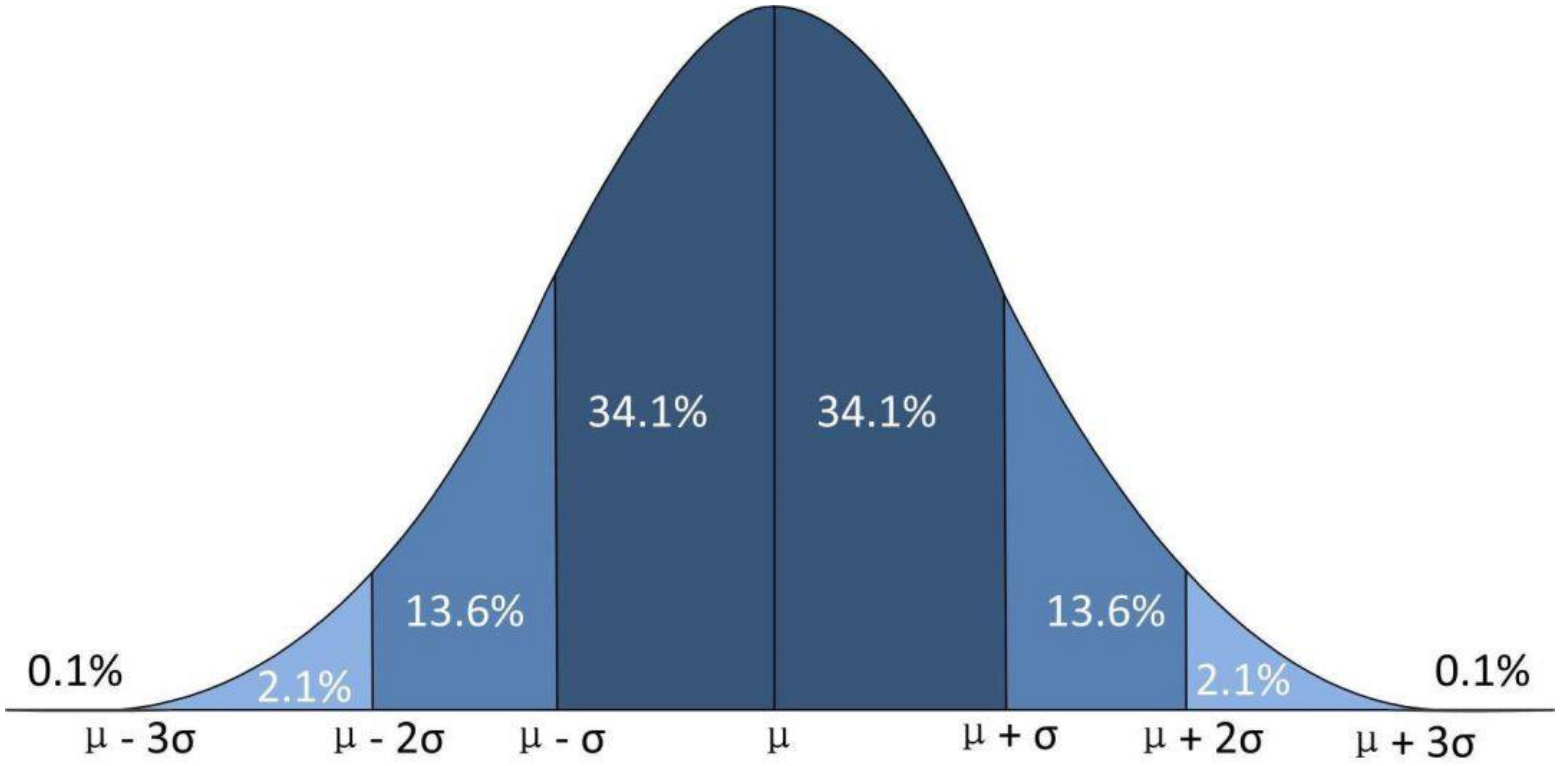


数据清理：异常值检测

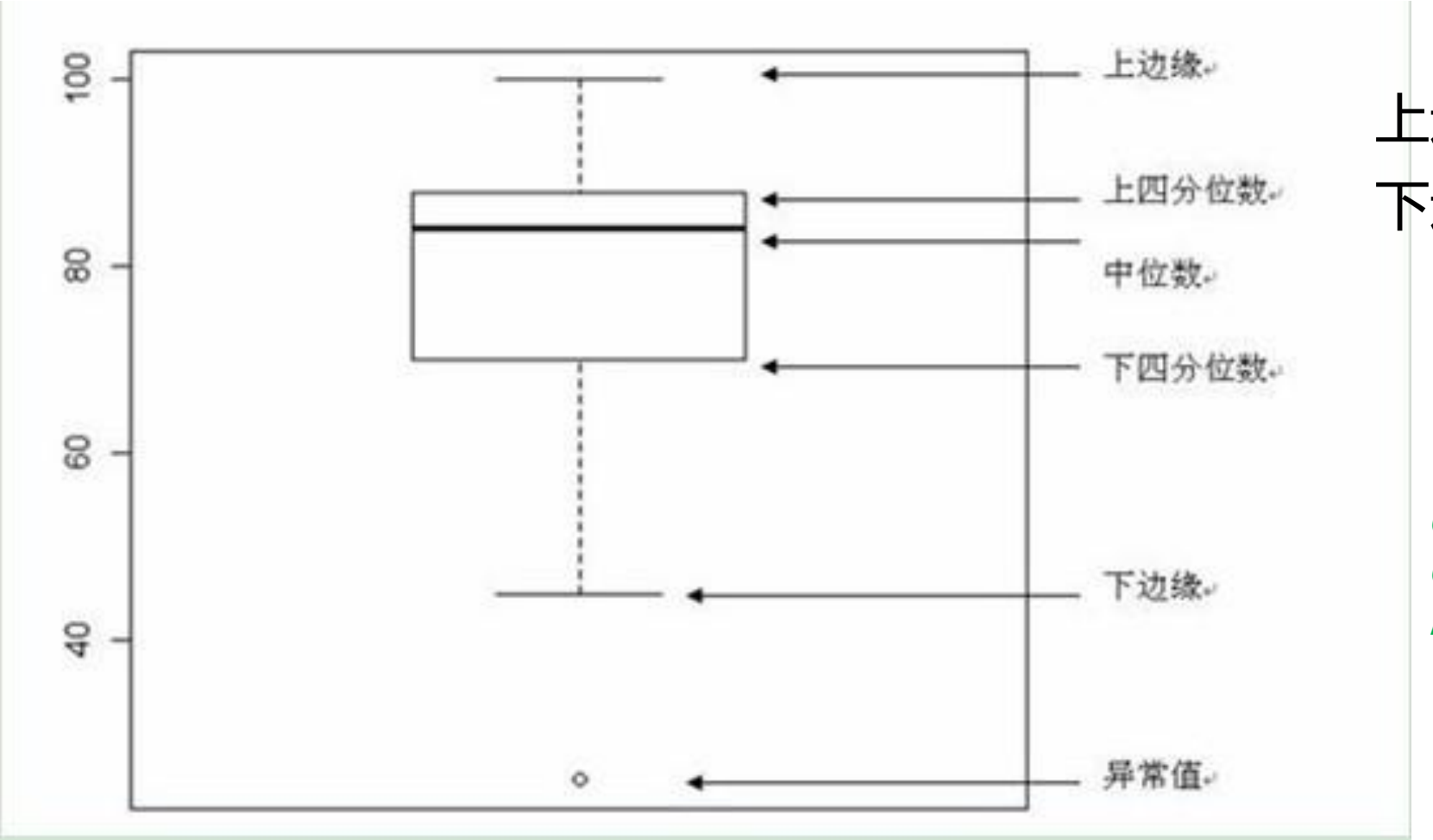
异常值的常用检测方法：

- 根据数据的具体含义确定正常值的范围，不在范围内的视为异常值
- 按照普遍数据分布规律或数据自身的分布规律判断，如对于正态分布，应用3σ准则检测异常值
- 使用箱形图检测异常值

数据清理：异常值检测——3σ原则



数据清理：异常值检测——箱形图



上边缘 =Q3 + kIQR

下边缘 =Q1 - kIQR

Q3：上四分位数
Q1：下四分位数
IQR(四分位距）=Q3-Q1

数据清理：异常值检测——箱形图

```
DataFrame.boxplot(column=None, by=None, ax=None, fontsize=None, rot=0, grid=True,
figsize=None, layout=None, return_type=None, backend=None, **kwargs)        [source]
```

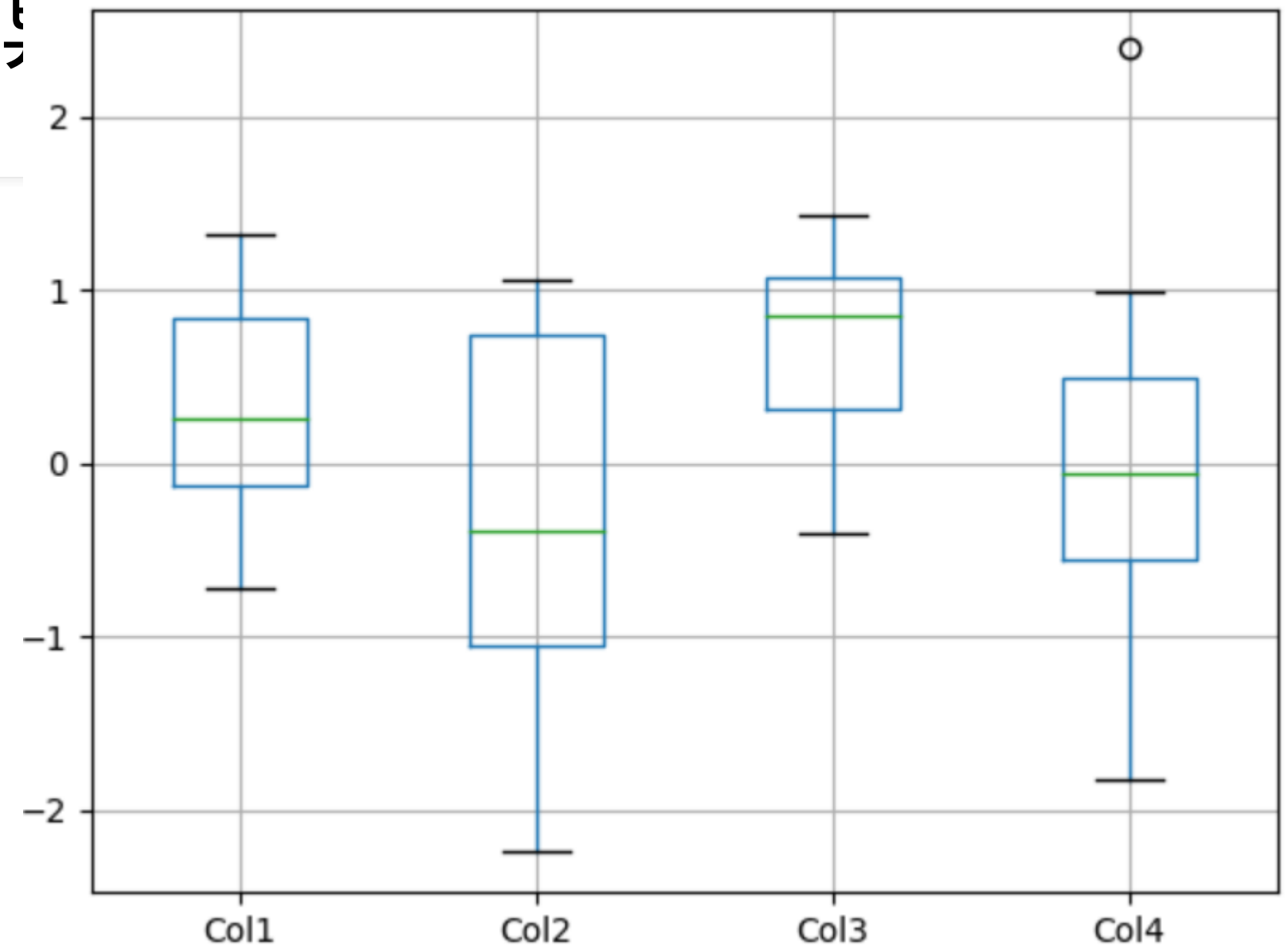
- column：被检测的列名
- fontsize：箱型图坐标轴的字体大小
- rot：箱型图坐标轴的旋转角度
- grid：箱型图窗口的大小
- return_type：返回的对象类型
  - ‘axes’：返回绘制箱型图的绘图区域，为默认值
  - ‘dict’：返回一个字典，其值为箱型图线条matplotlib的Line对象
  - ‘both’ :返回一个包含上述两个对象的元组

数据清理：异常值检测——箱形图

```
>>> np.random.seed(1234)
>>> df = pd.DataFrame(np.random.randn(10,4),
...                   columns=['Col1','Col2','Col3','Col4'])
>>> print(df)
       Col1      Col2      Col3      Col4
0  0.471435 -1.190976  1.432707 -0.312652
1 -0.720589  0.887163  0.859588 -0.636524
2  0.015696 -2.242685  1.150036  0.991946
3  0.953324 -2.021255 -0.334077  0.002118
4  0.405453  0.289092  1.321158 -1.546906
5 -0.202646 -0.655969  0.193421  0.553439
6  1.318152 -0.469305  0.675554 -1.817027
7 -0.183109  1.058969 -0.397840  0.337438
8  1.047579  1.045938  0.863717 -0.122092
9  0.124713 -0.322795  0.841675  2.390961
```

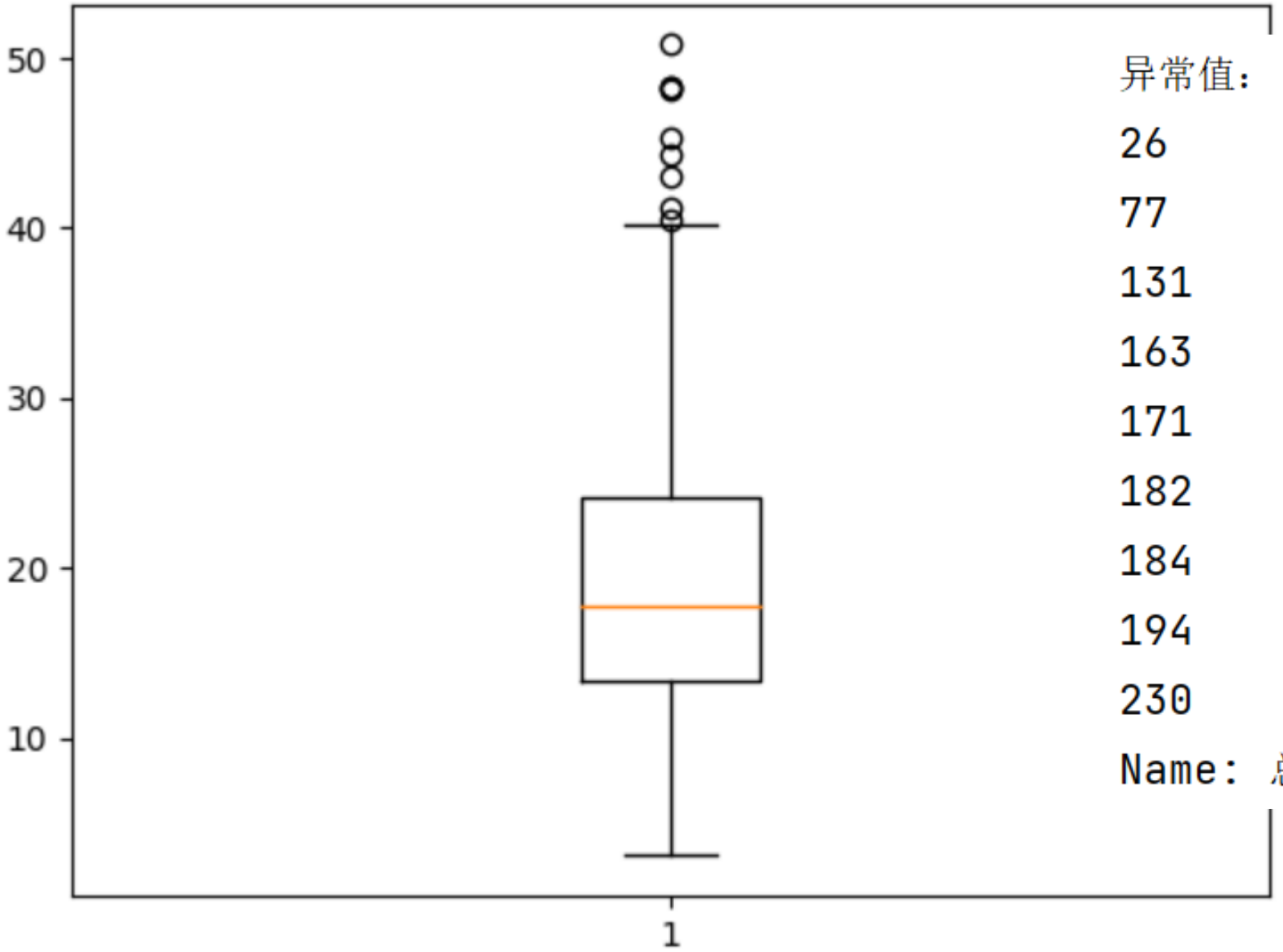
数据清理：

```
>>> df.boxplot()
```



数据清理：异常值检测——箱形图

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_excel('tips.xlsx')
plt.boxplot(x=df['总消费'])
plt.show()
q1 = df['总消费'].quantile(q=0.25)
q3 = df['总消费'].quantile(q=0.75)
low_limit = q1-1.5*(q3-q1)
high_limit = q3+1.5*(q3-q1)
val = df['总消费'][(df['总消费']>high_limit)|(df['总消费']<low_limit) ]
print('异常值：')
print(val)
```




```
异常值:
26     44.30
77     43.11
131    48.27
163    48.17
171    50.81
182    45.35
184    40.55
194    48.33
230    41.19
Name: 总消费, dtype: float64
```

数据清理：异常值的处理

对于检测出的异常值，需要进一步分析其出现原因及是否为真正的异常值。根据分析结果对异常值进行处理：

- 保留异常值
- 删除异常值
- 替换异常值

数据清理：删除异常值

- 使用Pandas删除数据的drop()方法

```
df.drop(labels=None, axis=0, index=None, columns=None, level
=None, inplace=False, errors='raise')
```

- 将异常值替换为NaN，然后使用Pandas删除缺失值的方法

数据清理：删除异常值

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_excel('tips.xlsx')
q1 = df['总消费'].quantile(q=0.25)
q3 = df['总消费'].quantile(q=0.75)
low_limit = q1-1.5*(q3-q1)
high_limit = q3+1.5*(q3-q1)
df.drop(df[(df['总消费']>high_limit)|(df['总消费']<low_limit)].index,inplace=True)
val = df['总消费'][(df['总消费']>high_limit)|(df['总消费']<low_limit) ]
print('异常值：')
print(val)
```

```
异常值：
Series([], Name: 总消费, dtype: float64)
```

数据清理：替换异常值

- 使用Pandas替换数据的replace()方法

```
df.replace(to_replace=None, value=None, inplace=False, limit=
None, regex=False, method='pad')
```

- 将异常值替换为NaN，然后使用Pandas替换或插补缺失值的方法

数据清理：替换异常值

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_excel('tips.xlsx')
q1 = df['总消费'].quantile(q=0.25)
q2 = df['总消费'].quantile(q=0.5)
q3 = df['总消费'].quantile(q=0.75)
low_limit = q1-1.5*(q3-q1)
high_limit = q3+1.5*(q3-q1)
val = df['总消费'][(df['总消费']>high_limit)|(df['总消费']<low_limit) ]
for va in val.values:
    df.replace(va,q2,inplace=True)
val = df['总消费'][(df['总消费']>high_limit)|(df['总消费']<low_limit) ]
print('异常值：')
print(val)
```

```
异常值：
Series([], Name: 总消费, dtype: float64)
```

数据预处理方法

- 数据清理
- 数据变换
- 数据规约
- 数据集成
- 数据格式转换

数据变换操作

- 数据标准化
- 数据离散化
- 数据泛化

数据标准化（Normalization）

- 数据标准化处理是将数据按照一定的规则缩放，使数据映射到一个较小的特定空间
- 数据标准化处理主要包括数据趋同化处理和无量纲化处理两个方面，目的在于避免不同性质的数据或不同的数据量级对统计分析造成影响

数据标准化：最小—最大标准化

最小—最大标准化（Min-Max Normalization）对数据进行线性变换，使数据范围变为[0,1]

$$x' = \frac{x - min}{max - min}$$

df_norm = (df – df.min())/(df.max()-df.min())

数据标准化：最小—最大标准化

```
>>> np.random.seed(1)
>>> df = pd.DataFrame(np.random.randn(4,4)*3+4)
>>> print(df)
          0         1         2         3
0  8.873036  2.164731  2.415485  0
1  6.596223 -2.904616  9.234435  1
2  4.957117  3.251889  8.386324 -2
3  3.032748  2.847837  7.401308  0
```

```
>>> df_norm = (df-df.min())/(df.max()-df.min())
>>> print(df_norm)
          0         1         2         3
0  1.000000  0.823413  0.000000  0.759986
1  0.610154  0.000000  1.000000  1.000000
2  0.329499  1.000000  0.875624  0.000000
3  0.000000  0.934370  0.731172  0.739260
```

数据标准化：最小—最大标准化

```
>>> df.min()
0    3.032748
1   -2.904616
2    2.415485
3   -2.180422
dtype: float64
>>> df.max()
0    8.873036
1    3.251889
2    9.234435
3    1.716379
dtype: float64
```

```
>>> df_norm.min()
0    0.0
1    0.0
2    0.0
3    0.0
dtype: float64
>>> df_norm.max()
0    1.0
1    1.0
2    1.0
3    1.0
dtype: float64
```

数据标准化：均值标准化

均值标准化又称z-score标准化、标准差标准化，将数据转化为标准正态分布（均值为0，标准差为1）

$$x' = \frac{x - \mu}{\sigma}$$

df_norm = (df – df.mean())/df.std()

数据标准化：均值标准化

```
>>> np.random.seed(1)
>>> df = pd.DataFrame(np.random.randn(4,4)*3+4)
>>> print(df)
          0         1         2
0  8.873036  2.164731  2.415485  0.
1  6.596223 -2.904616  9.234435  1.
2  4.957117  3.251889  8.386324 -2.
3  3.032748  2.847837  7.401308  0.
```

```
>>> df_norm2 = (df-df.mean())/df.std()
>>> print(df_norm2)
          0         1         2         3
0  1.213741  0.287871 -1.454237  0.312166
1  0.295115 -1.481492  0.777218  0.866440
2 -0.366215  0.667324  0.499679 -1.442906
3 -1.142640  0.526297  0.177340  0.264301
```

数据标准化：均值标准化

```
>>> df.mean()
0    5.864781
1    1.339960
2    6.859388
3    0.254344
dtype: float64
>>> df.std()
0    2.478499
1    2.865068
2    3.055832
3    1.687404
dtype: float64
```

```
>>> df_norm2.mean()
0   -5.551115e-17
1    1.110223e-16
2    5.551115e-17
3    4.163336e-17
dtype: float64
>>> df_norm2.std()
0    1.0
1    1.0
2    1.0
3    1.0
dtype: float64
```

数据离散化

- 数据离散化是在数据的取值范围内设定若干个离散的划分点，将取值范围划分为若干个离散化区间，分别用不同的符号或整数值代表落在每个子区间的数据
- 离散化的原因包括：减少数据量以提高计算效率，离散化的特征更易于理解，有些模型只能使用离散化的数据，使模型结果更加稳定等

数据离散化

空气质量指数	空气质量等级	空气质量指数类别及其表示颜色		对健康影响情况	建议采取的措施
0~50	一级	优	绿色	空气质量令人满意，基本无空气污染	各类人群可正常活动
51~100	二级	良	黄色	空气质量可接受，但某些污染物可能对极少数异常敏感人群健康有较弱影响	极少数异常敏感人群应减少户外活动
101~150	三级	轻度污染	橙色	易感人群症状有轻度加剧，健康人群出现刺激症状	儿童、老年人及心脏病、呼吸系统疾病患者应减少长时间、高强度的户外锻炼
151~200	四级	中度污染	红色	进一步加剧易感人群症状，可能对健康人群心脏、呼吸系统有影响	儿童、老年人及心脏病、呼吸系统疾病患者避免长时间、高强度的户外锻炼，一般人群适量减少户外运动
201~300	五级	重度污染	紫色	心脏病和肺病患者症状显著加剧运动耐受力降低，健康人群普遍出现症状	儿童、老年人和心脏病、肺病患者应停留在室内，停止户外运动，一般人群减少户外运动
>300	六级	严重污染	褐红色	健康人群运动耐受力降低，有明显强烈症状，提前出现某些疾病	儿童、老年人和病人应当留在室内避免体力消耗，一般人群应避免户外活动

数据离散化：面元划分（分箱）

- 面元划分：数据被离散化处理，按照一定的映射关系划分为相应的面元（区间）

序列	age
0	19
1	46
2	25
3	55
4	30
5	45
6	52

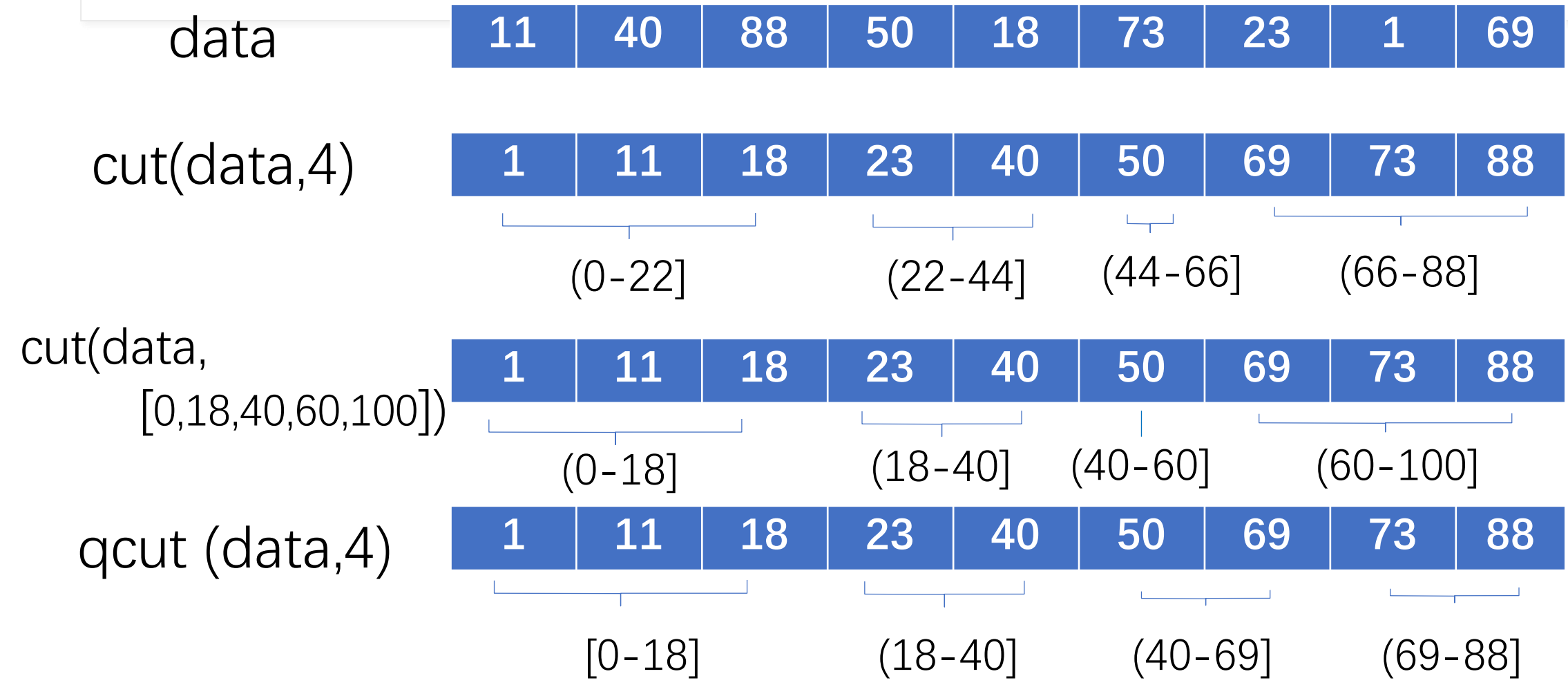
面元划分

序列	age
0	(18, 30]
1	(30, 50]
2	(18, 30]
3	(50, 100]
4	(18, 30]
5	(30, 50]
6	(50, 100]

数据离散化

- 数据离散化处理的两种常见方法：
  - ✓ 按值划分（等宽法）：将数据的取值范围从小到大划分成具有相同宽度的区间，或指定每个区间的起始数据值
  - ✓ 按个数划分（等频法）：将相同数量的数据划分到每个区间

数据离散化: cut和qcut方法



数据离散化：按值划分

pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3, include_lowest=False, duplicates='raise', ordered=True)

- x：需要离散化的数据
- bins：划分面元的依据，为整数、序列尺度或间隔索引
- right：为True表示区间右边闭合，False表示区间左边闭合
- labels：分组的自定义标签
- retbins：是否返回面元
- precision：存储或展示标签的精度，默认为3（小数点后3位）
- include_lowest：是否包含区间的左端点

数据离散化：按值划分

```
>>> ages = pd.Series([18,22,5,76,43,9,16,35,66,41,29])
>>> bins = 4
>>> cuts = pd.cut(ages, bins)
>>> cuts
0     (4.929, 22.75]
1     (4.929, 22.75]
2     (4.929, 22.75]
3      (58.25, 76.0]
4      (40.5, 58.25]
5     (4.929, 22.75]
6     (4.929, 22.75]
7      (22.75, 40.5]
8      (58.25, 76.0]
9      (40.5, 58.25]
10     (22.75, 40.5]
dtype: category
Categories (4, interval[float64, right]): [(4.929, 22.75] < (22.75, 40.5] < (40.5, 58.25] <
                                           (58.25, 76.0]]
```

```
>>> pd.value_counts(cuts)
(4.929, 22.75]    5
(22.75, 40.5]     2
(40.5, 58.25]     2
(58.25, 76.0]     2
dtype: int64
```

数据离散化：按值划分

```
>>> bins = [0,18,40,60,100]
>>> cuts = pd.cut(ages, bins)
>>> cuts
0        (0, 18]
1       (18, 40]
2        (0, 18]
3      (60, 100]
4       (40, 60]
5        (0, 18]
6        (0, 18]
7       (18, 40]
8      (60, 100]
9       (40, 60]
10      (18, 40]
dtype: category
Categories (4, interval[int64, right]): [(0, 18] < (18, 40] < (40, 60] < (60, 100]]
```

```
>>> pd.value_counts(cuts)
(0, 18]      4
(18, 40]     3
(40, 60]     2
(60, 100]    2
dtype: int64
```

数据离散化：按个数划分

pandas.qcut(x, q, labels=None, retbins=False, precision=3, duplicates='raise')

- x：需要离散化的数据
- q：划分面元的数量
- labels：分组的自定义标签
- retbins：是否返回面元
- precision：存储或展示标签的精度，默认为3（小数点后3位）
- duplicates：如果面元的边缘不唯一，则抛出错误（‘raise’）或删除非唯一数据（‘drop’）

数据离散化：按个数划分

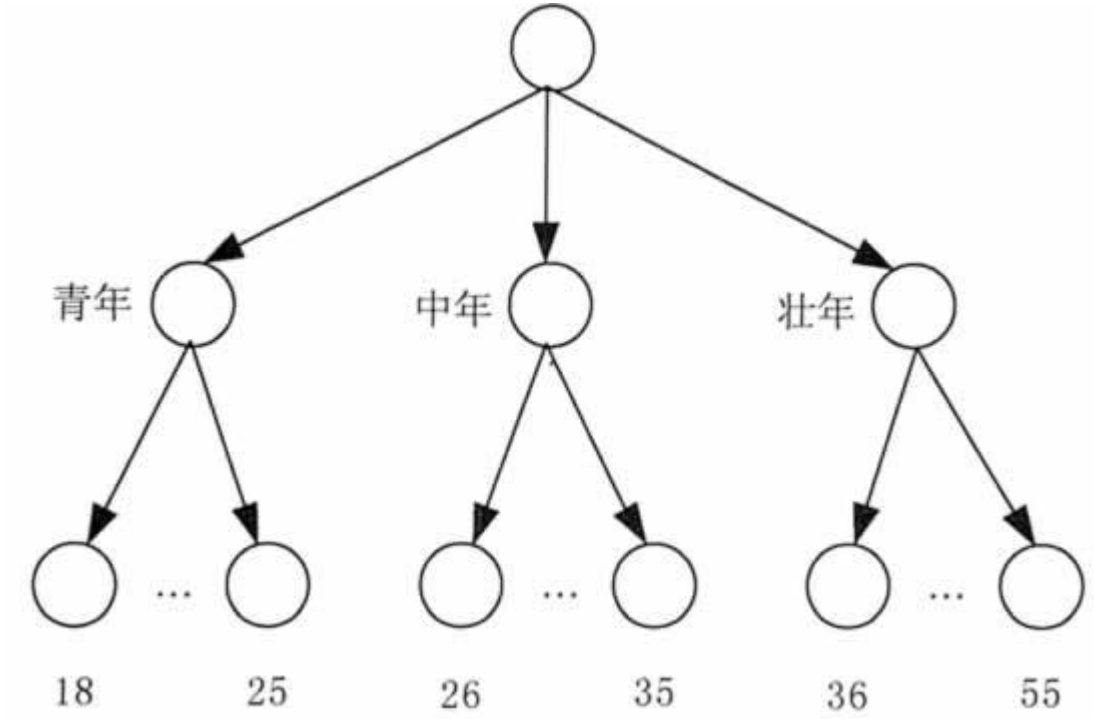
```
>>> cuts = pd.qcut(ages, q)  #q = 4
>>> cuts
0       (17.0, 29.0]
1       (17.0, 29.0]
2      (4.999, 17.0]
3       (42.0, 76.0]
4       (42.0, 76.0]
5      (4.999, 17.0]
6      (4.999, 17.0]
7       (29.0, 42.0]
8       (42.0, 76.0]
9       (29.0, 42.0]
10      (17.0, 29.0]
dtype: category
Categories (4, interval[float64, right]): [(4.999, 17.0] < (17.0, 29.0] < (29.0, 42.0] <
                                           (42.0, 76.0]]
```

```
>>> pd.value_counts(cuts)
(4.999, 17.0]    3
(17.0, 29.0]     3
(42.0, 76.0]     3
(29.0, 42.0]     2
dtype: int64
```

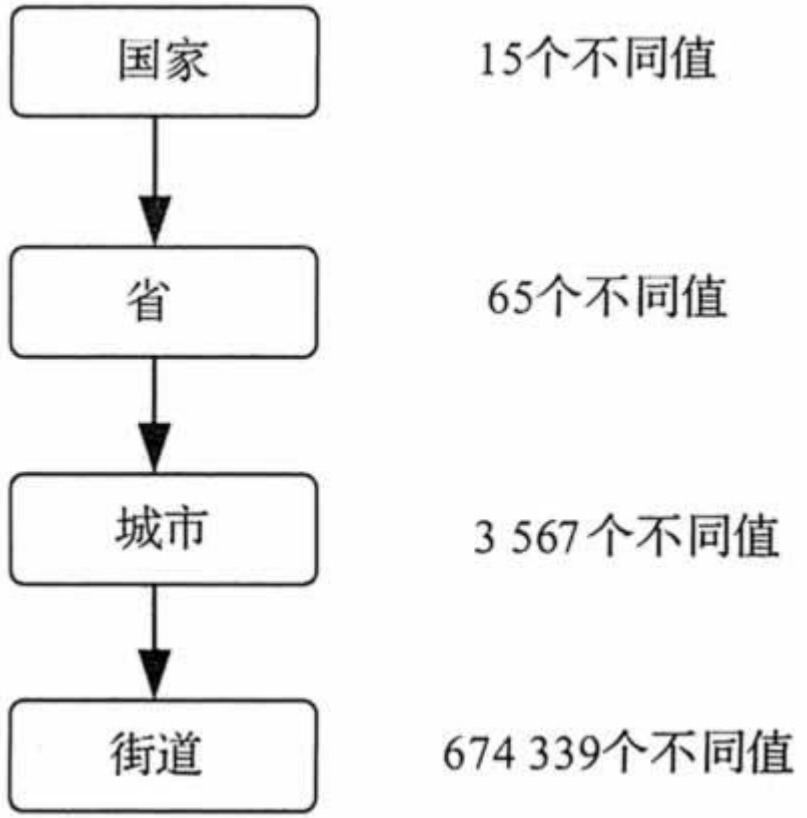
数据泛化

- 数据泛化处理是用更抽象(更高层次)的概念来取代低层次或数据层的数据对象
- 数据泛化可用于对给定数据集的简洁汇总
- 在数据挖掘中用于概念描述（描述式数据挖掘）

数据泛化：概念层次树



数值概念层次树



属性概念层次树

数据预处理方法

- 数据清理
- 数据变换
- 数据规约
- 数据集成
- 数据格式转换

数据规约

- 数据规约类似于数据集的压缩，其主要作用是从原有数据集中获得一个精简的数据集，以提高数据分析或数据挖掘的工作效率
- 数据规约在减低数据规模的基础上，应保留原有数据集的完整性，保证使用精简的数据集进行数据分析或数据挖掘的结果与使用原有数据集获得的结果基本相同

数据规约的方法

- 维度规约：减少所需属性的数目
- 数量规约：用较小规模的数据替换或估计原数据，主要方法包括回归与线性对数模型、直方图、聚类、采样和数据立方体
- 数据压缩：利用编码或转换将原有数据集压缩为一个较小规模的数据集

数据集成

- 数据集成即将多个来源的数据合并为一个统一的数据源的过程
- 数据集成期间可能出现实体矛盾、冗余属性和元组重复等问题，需要在集成后进行数据清理

数据集成：合并数据的函数或方法

	函数/方法	描述
主键合并数据	merge()	根据一个或多个键连接两组数据
	merge_ordered()	通过可选的填充值/差值连接两组有序的数据
	merge_asof()	根据匹配最近的键连接两个DataFrame对象（必须按键进行排序）
堆叠合并数据	join()	根据行索引连接多组数据
	concat()	沿着某一轴方向堆叠多组数据
	append()	向数据末尾追加若干行数据
重叠合并数据	combine_first()	使用一个对象填充另一个对象中相同位置的缺失值

数据预处理方法

- 数据清理
- 数据变换
- 数据规约
- 数据集成
- 数据格式转换

数据格式化：设置小数位数

df.round(decimals=0, *args, **kwargs)

- decimals：每一列四舍五入的小数位数，为整型、字典或Series对象。如果是整数，则将每一列四舍五入到相同的位置；否则，将dict和Series舍入到可变类型的位置。如果小数是类似于字典的，则列名应该在键中；如果小数是级数，列名应该在索引中。没有包含在小数中的任何列都将保持原样，非输入列的小数元素将被忽略
- args, kwargs：附加的关键字参数

数据格式化：设置小数位数

```
>>> import numpy as np
>>> df = pd.DataFrame(np.random.random([5,5]),)
>>> df = pd.DataFrame(np.random.random([5,5]),
...                   columns=['A1','A2','A3','A4','A5'])
>>> print(df)
         A1        A2        A3        A4        A5
0  0.173336  0.185013  0.842482  0.940283  0.044306
1  0.078331  0.704547  0.631699  0.217729  0.295444
2  0.957878  0.671661  0.110745  0.745400  0.820567
3  0.382970  0.120367  0.969469  0.774557  0.877386
4  0.993134  0.842031  0.218594  0.304956  0.355373
```

数据格式化：设置小数位数

```
>>> print(df.round(2))
     A1    A2    A3    A4    A5
0  0.17  0.19  0.84  0.94  0.04
1  0.08  0.70  0.63  0.22  0.30
2  0.96  0.67  0.11  0.75  0.82
3  0.38  0.12  0.97  0.77  0.88
4  0.99  0.84  0.22  0.30  0.36
```

```
>>> print(df.round({'A1':1,'A2':2}))
    A1    A2        A3        A4        A5
0  0.2  0.19  0.842482  0.940283  0.044306
1  0.1  0.70  0.631699  0.217729  0.295444
2  1.0  0.67  0.110745  0.745400  0.820567
3  0.4  0.12  0.969469  0.774557  0.877386
4  1.0  0.84  0.218594  0.304956  0.355373
```

数据格式化：apply函数

df.apply(func, axis=0, raw=False, result_type=None, args=(), **kwargs)

- func：对所遍历元素执行的函数
- axis: 执行func的轴
- raw: 行/列作为Series（False）还是作为ndarray（True）传递给函数
- result_type：返回值的类型，仅当axis=1时应用：
  - ✓ ‘expand’：类似于列表的结果转换为列
  - ✓ ‘reduce’：如果可能，返回一个Series对象而非展开为列
  - ✓ ‘broadcast’：将结果广播为DataFrame的原始形状，保留原始的索引和列
  - ✓ ‘None’：返回值类型取决于函数的返回值。为默认值

数据格式化：apply函数

Serial.apply(func, convert_dtype=True, args=(), **kwargs)

- func：对所遍历元素执行的函数
- convert_dtype: 是否尝试为函数执行结果找到更好的dtype，默认为True

数据格式化：设置百分比

```
>>> df['A1']=df['A1'].apply(lambda x:format(x,'.2%'))
```

```
>>> print(df)
       A1        A2        A3        A4        A5
0  17.33%  0.185013  0.842482  0.940283  0.044306
1   7.83%  0.704547  0.631699  0.217729  0.295444
2  95.79%  0.671661  0.110745  0.745400  0.820567
3  38.30%  0.120367  0.969469  0.774557  0.877386
4  99.31%  0.842031  0.218594  0.304956  0.355373
```

数据格式化：设置百分比

```
>>> df['百分比']=df['A1'].apply(lambda x:format(x,'.0%'))
>>> print(df)
         A1        A2        A3        A4        A5  百分比
0  0.173336  0.185013  0.842482  0.940283  0.044306  17%
1  0.078331  0.704547  0.631699  0.217729  0.295444   8%
2  0.957878  0.671661  0.110745  0.745400  0.820567  96%
3  0.382970  0.120367  0.969469  0.774557  0.877386  38%
4  0.993134  0.842031  0.218594  0.304956  0.355373  99%
```

数据行列转换：一列数据转换为多列数据

```
>>> df = pd.read_json('weather.json',lines=True)
>>> print(df)
```

	date	high_temp	low_temp	weather	wind
0	2021-08-01 星期日	32℃	22℃	晴	北风 3级
1	2021-08-02 星期一	32℃	23℃	多云	东北风 2级
2	2021-08-03 星期二	31℃	24℃	阴	南风 2级
3	2021-08-04 星期三	34℃	26℃	多云	南风 2级
4	2021-08-05 星期四	32℃	23℃	多云	东南风 2级
5	2021-08-06 星期五	33℃	22℃	多云	南风 2级
6	2021-08-07 星期六	33℃	23℃	晴	东南风 2级
7	2021-08-08 星期日	32℃	23℃	多云	南风 2级
8	2021-08-09 星期一	30℃	22℃	晴	东风 2级

数据行列转换：一列数据转换为多列数据

Serial.str.split (pat=None, n=-1, expand=False)

- pat：字符串、符号或正则表达式，表示字符串分割的依据。默认以空格分割字符串
- n: 分割次数，默认为-1。0或-1都将对字符串进行完全拆分
- expand: 分割后的结果是否转换为DataFrame格式，默认为False

数据行列转换：一列数据转换为多列数据

```
>>> series=df['date'].str.split(' ',expand=True)
>>> df['日期']=series[0]
>>> df['星期']=seires[1]
```

```
>>> print(df.head())
         date high_temp low_temp weather    wind          日期    星期
0  2021-08-01 星期日       32℃      22℃      晴   北风 3级  2021-08-01  星期日
1  2021-08-02 星期一       32℃      23℃     多云  东北风 2级  2021-08-02  星期一
2  2021-08-03 星期二       31℃      24℃      阴   南风 2级  2021-08-03  星期二
3  2021-08-04 星期三       34℃      26℃     多云   南风 2级  2021-08-04  星期三
4  2021-08-05 星期四       32℃      23℃     多云  东南风 2级  2021-08-05  星期四
```

内容

- 数据导入
- 数据观察
- 数据预处理
- 数据导出

数据导出：导出为.csv文件

pandas.DataFrame.to_csv

```
DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='', float_format=None, columns=None,
header=True, index=True, index_label=None, mode='w', encoding=None, compression='infer',
quoting=None, quotechar='"', line_terminator=None, chunksize=None, date_format=None,
doublequote=True, escapechar=None, decimal='.', errors='strict', storage_options=None)
```

to_csv方法的常用功能

指定分隔符

df.to_csv（‘Result.csv’，sep=‘?’）

替换空值

df.to_csv（‘Result.csv’，na_rep=‘NA’）

格式化数据

df.to_csv（‘Result.csv’，float_format=‘%2.f’）

to_csv方法的常用功能

导出指定列

```
df.to_csv( ‘Result.csv’ , columns = [ ‘name’ ])
```

不导出列名

```
df.to_csv( ‘Result.csv’ , header = False)
```

不导出行索引

```
df.to_csv( ‘Result.csv’ , index = False)
```

数据导出：导出为.xlsx文件

pandas.DataFrame.to_excel

```
DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep='', float_format=None,
columns=None, header=True, index=True, index_label=None, startrow=0, startcol=0,
engine=None, merge_cells=True, encoding=None, inf_rep='inf', verbose=True,
freeze_panes=None, storage_options=None)                                    [source]
```

数据预处理：总结

- 数据预处理的目的是将数据转换为准确、完整、简洁的高质量数据，以更好地服务于数据分析、数据挖掘或机器学习等工作
- 数据预处理一般包括数据清理、数据变换、数据集成、数据规约等步骤，各项步骤没有明确的执行顺序或执行次数
- 在一般的数据预处理步骤之外，根据数据的具体情况和要进行的数据分析工作，还需要进行数据格式化、数据分列等预处理工作