数据预处理

数据分析的基本流程

分析的问题是什么？
你想得出哪些结论？

问题定义

网络爬虫

互联网数据

内部数据
外部数据

数据获取

数据预处理

数据清洗，集成，加工

数据分析

探究内部关系，进行分析预测

统计量的描述和展示

数据可视化及报告撰写

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=None, 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=None, 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： world_population.csv

Rank	CCA3	Country	Capital	Continent	2022 Population	2020 Popu	2015 Popu	2010 Popu	2000 Popu	1990 Popu	1980 Popu	1970 Popu	Area (km	Density (p	Growth Ra	World Population Percentage
36	AFG	Afghanista	Kabul	Asia	41128771	38972230	33753499	28189672	19542982	10694796	12486631	10752971	652230	63.0587	1.0257	0.52
138	ALB	Albania	Tirana	Europe	2842321	2866849	2882481	2913399	3182021	3295066	2941651	2324731	28748	98.8702	0.9957	0.04
34	DZA	Algeria	Algiers	Africa	44903225	43451666	39543154	35856344	30774621	25518074	18739378	13795915	2381741	18.8531	1.0164	0.56
213	ASM	American	Pago Pag	Oceania	44273	46189	51368	54849	58230	47818	32886	27075	199	222.4774	0.9831	0
203	AND	Andorra	Andorra la	Europe	79824	77700	71746	71519	66097	53569	35611	19860	468	170.5641	1.01	0
42	AGO	Angola	Luanda	Africa	35588987	33428485	28127721	23364185	16394062	11828638	8330047	6029700	1246700	28.5466	1.0315	0.45
224	AIA	Anguilla	The Valley	North Am	15857	15585	14525	13172	11047	8316	6560	6283	91	174.2527	1.0066	0
201	ATG	Antigua a	Saint Johr	North Am	93763	92664	89941	85695	75055	63328	64888	64516	442	212.1335	1.0058	0
33	ARG	Argentina	Buenos Ai	South Am	45510318	45036032	43257065	41100123	37070774	32637657	28024803	23842803	2780400	16.3683	1.0052	0.57
140	ARM	Armenia	Yerevan	Asia	2780469	2805608	2878595	2946293	3168523	3556539	3135123	2534377	29743	93.4831	0.9962	0.03
198	ABW	Aruba	Oranjesta	North Am	106445	106585	104257	100341	89101	65712	62267	59106	180	591.3611	0.9991	0
55	AUS	Australia	Canberra	Oceania	26177413	25670051	23820236	22019168	19017963	17048003	14706322	12595034	7692024	3.4032	1.0099	0.33
99	AUT	Austria	Vienna	Europe	8939617	8907777	8642421	8362829	8010428	7678729	7547561	7465301	83871	106.5877	1.002	0.11
91	AZE	Azerbaijar	Baku	Asia	10358074	10284951	9863480	9237202	8190337	7427836	6383060	5425317	86600	119.6082	1.0044	0.13
176	BHS	Bahamas	Nassau	North Am	409984	406471	392697	373272	325014	270679	223752	179129	13943	29.4043	1.0051	0.01
154	BHR	Bahrain	Manama	Asia	1472233	1477469	1362142	1213645	711442	517418	362595	222555	765	1924.488	1.0061	0.02
8	BGD	Banglades	Dhaka	Asia	171186372	1.67E+08	1.58E+08	1.48E+08	1.29E+08	1.07E+08	83929765	67541860	147570	1160.035	1.0108	2.15
186	BRB	Barbados	Bridgetow	North Am	281635	280693	278083	274711	264657	258868	253575	241397	430	654.9651	1.0015	0
96	BLR	Belarus	Minsk	Europe	9534954	9633740	9700609	9731427	10256483	10428525	9817257	9170786	207600	45.9295	0.9955	0.12
81	BEL	Belgium	Brussels	Europe	11655930	11561717	11248303	10877947	10264343	9959560	9828986	9629376	30528	381.8111	1.0038	0.15
177	BLZ	Belize	Belmopan	North Am	405272	394921	359871	322106	240406	182589	145133	120905	22966	17.6466	1.0131	0.01
77	BEN	Benin	Porto-No	Africa	13352864	12643123	10932783	9445710	6998023	5133419	3833939	3023443	112622	118.5635	1.0274	0.17
206	BMU	Bermuda	Hamilton	North Am	64184	64031	63144	63447	61371	57470	53565	52019	54	1188.593	1	0
165	BTN	Bhutan	Thimphu	Asia	782455	772506	743274	705516	587207	558442	415257	298894	38394	20.3796	1.0064	0.01
80	BOL	Bolivia	Sucre	South Am	12224110	11936162	11090085	10223270	8592656	7096194	5736088	4585693	1098581	11.1272	1.012	0.15
137	BIH	Bosnia an	Sarajevo	Europe	3233526	3318407	3524324	3811088	4179350	4494310	4199820	3815561	51209	63.1437	0.9886	0.04
144	BWA	Botswana	Gaborone	Africa	2630296	2546402	2305171	2091664	1726985	1341474	938578	592244	582000	4.5194	1.0162	0.03
7	BRA	Brazil	Brasilia	South Am	215313498	2.13E+08	2.05E+08	1.96E+08	1.76E+08	1.51E+08	1.22E+08	96369875	8515767	25.2841	1.0046	2.7
221	VGB	British Vir	Road Tow	North Am	31305	30910	29366	27556	20104	15617	11109	9581	151	207.3179	1.0059	0

数据导入：导入.csv文件

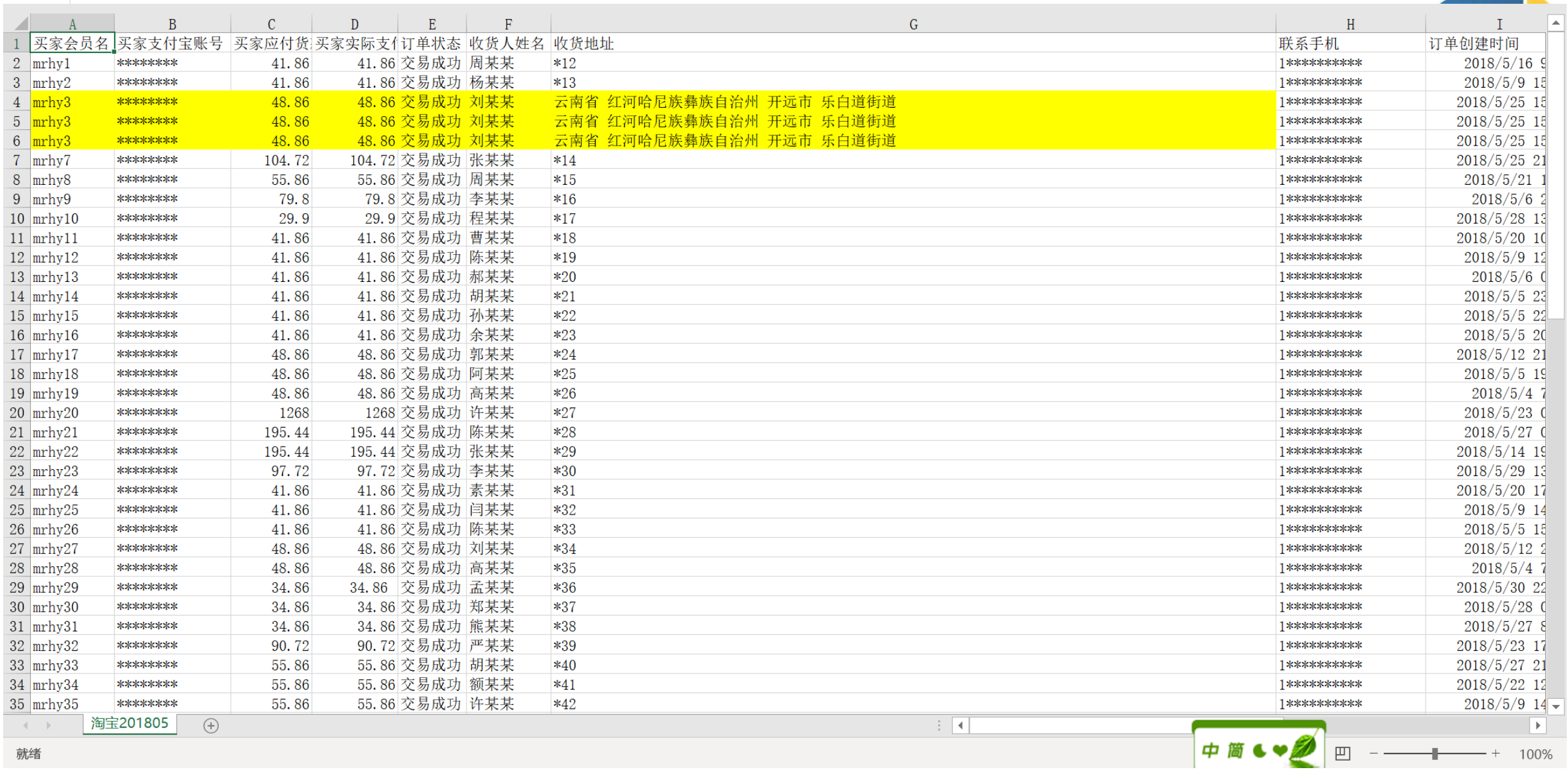
```
import pandas as pd
#设置数据显示的最大列数和宽度
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
#设置float数据显示格式，避免以科学计数法显示
pd.set_option('display.float_format', lambda x: '%.2f' % x)
population = pd.read_csv('world_population.csv')
```

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=None, 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, decimal='.', 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')
```

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(population.head(10))
   Rank CCA3             Country          Capital      Continent  2022 Population  2020 Pop
0    36  AFG         Afghanistan            Kabul           Asia          41128771         3
1   138  ALB             Albania           Tirana         Europe           2842321
2    34  DZA             Algeria          Algiers         Africa          44903225         4
3   213  ASM      American Samoa        Pago Pago        Oceania             44273
4   203  AND             Andorra  Andorra la Vella         Europe             79824
5    42  AGO              Angola           Luanda         Africa          35588987         3
6   224  AIA            Anguilla       The Valley  North America             15857
7   201  ATG  Antigua and Barbuda     Saint John’s  North America             93763
8    33  ARG           Argentina     Buenos Aires  South America          45510318         4
9   140  ARM             Armenia           Yerevan           Asia           2780469
```

数据观察

df.tail(n)

```
>>> print(population.tail())
     Rank CCA3            Country   Capital Continent  2022 Population  2020 Popula
229   226  WLF  Wallis and Futuna  Mata-Utu   Oceania             11572          1
230   172  ESH     Western Sahara  El Aaiún    Africa            575986          55
231    46  YEM              Yemen     Sanaa      Asia          33696614        3228
232    63  ZMB             Zambia    Lusaka    Africa          20017675        1892
233    74  ZWE           Zimbabwe    Harare    Africa          16320537        1566
```

数据观察

df.sample(n)

```
>>> print(population.sample(8))
     Rank CCA3       Country      Capital      Continent  2022 Population  2020 Populat
223   133  URY       Uruguay   Montevideo  South America          3422794          3429
30    108  BGR      Bulgaria        Sofia         Europe          6781953          6979
108   129  KWT        Kuwait  Kuwait City           Asia          4268873          4360
227    51  VEN     Venezuela      Caracas  South America         28301696         28490
84     75  GIN        Guinea      Conakry         Africa         13859341         13205
187   113  SGP     Singapore    Singapore           Asia          5975689          5909
193    24  ZAF  South Africa     Pretoria         Africa         59893885         58801
136   169  MNE    Montenegro    Podgorica         Europe           627082           629
```

数据观察

```
>>> population.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Rank                         234 non-null    int64
 1   CCA3                         234 non-null    object
 2   Country                      234 non-null    object
 3   Capital                      234 non-null    object
 4   Continent                    234 non-null    object
 5   2022 Population              234 non-null    int64
 6   2020 Population              234 non-null    int64
 7   2015 Population              234 non-null    int64
 8   2010 Population              234 non-null    int64
 9   2000 Population              234 non-null    int64
 10  1990 Population              234 non-null    int64
 11  1980 Population              234 non-null    int64
 12  1970 Population              234 non-null    int64
 13  Area (km²)                   234 non-null    int64
 14  Density (per km²)            234 non-null    float64
 15  Growth Rate                  234 non-null    float64
 16  World Population Percentage  234 non-null    float64
dtypes: float64(3), int64(10), object(4)
memory usage: 31.2+ KB
```

数据观察

df.describe()

```
>>> population.describe()
        Rank  2022 Population  2020 Population  2015 Population  2010 Population  2000 Population  1990 Population  1980 Population  1970 Population  Area (km²)  De
count 234.00           234.00           234.00           234.00           234.00           234.00           234.00           234.00           234.00      234.00
mean  117.50      34074414.71      33501070.95      31729956.24      29845235.03      26269468.82      22710220.79      18984616.97      15786908.81   581449.38
std    67.69     136766424.80     135589876.92     130404992.75     124218487.63     111698206.72      97832173.35      81785186.08      67795091.64  1761840.86
min     1.00           510.00           520.00           564.00           596.00           651.00           700.00           733.00           752.00        1.00
25%    59.25        419738.50        415284.50        404676.00        393149.00        327242.00        264115.75        229614.25        155997.00     2650.00
50%   117.50       5559944.50       5493074.50       5307400.00       4942770.50       4292907.00       3825409.50       3141145.50       2604830.00    81199.50
75%   175.75      22476504.75      21447979.50      19730853.75      19159567.50      15762301.00      11869231.00       9826053.75       8817329.00   430425.75
max   234.00    1425887337.00    1424929781.00    1393715448.00    1348191368.00    1264099069.00    1153704252.00     982372466.00     822534450.00 17098242.00
```

数据观察

```
>>> population.iloc[:,[5,6,13,14,15]].describe()
       2022 Population  2020 Population  Area (km²)  Density (per km²)  Growth Rate
count           234.00           234.00      234.00             234.00       234.00
mean       34074414.71      33501070.95   581449.38             452.13         1.01
std       136766424.80     135589876.92  1761840.86            2066.12         0.01
min             510.00           520.00        1.00               0.03         0.91
25%          419738.50        415284.50     2650.00              38.42         1.00
50%         5559944.50       5493074.50    81199.50              95.35         1.01
75%        22476504.75      21447979.50   430425.75             238.93         1.02
max      1425887337.00    1424929781.00 17098242.00           23172.27         1.07
```

数据观察

```
>>> population.loc[population['Continent']=='Asia'].iloc[:,[5,6,13,14,15]].describe()
       2022 Population  2020 Population  Area (km²)  Density (per km²)  Growth Rate
count            50.00            50.00       50.00              50.00        50.00
mean       94427665.48      93261730.70   642762.82            1025.02         1.01
std       279720709.00     277537414.44  1495048.86            3535.09         0.01
min          449002.00        441725.00       30.00               2.17         0.98
25%         5309988.75       5180281.50    31355.50              71.51         1.00
50%        18082920.00      17688051.50   164302.50             122.97         1.01
75%        49985888.00      49522763.50   506865.00             336.35         1.01
max      1425887337.00    1424929781.00  9706961.00           23172.27         1.04
```

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：缺失值的检测

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

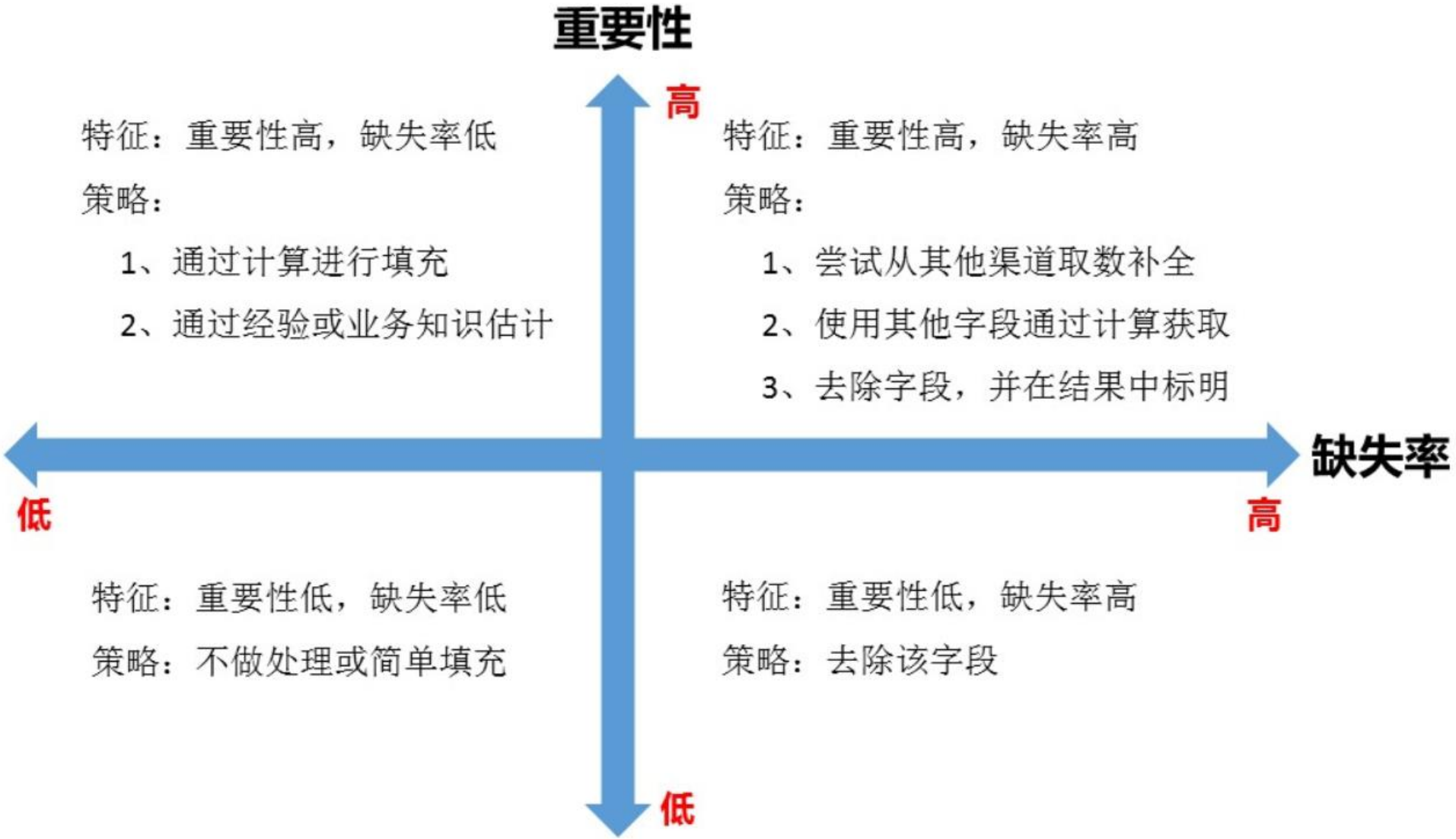
df.isnull(), df.isna()

df.notull(), df.notna()

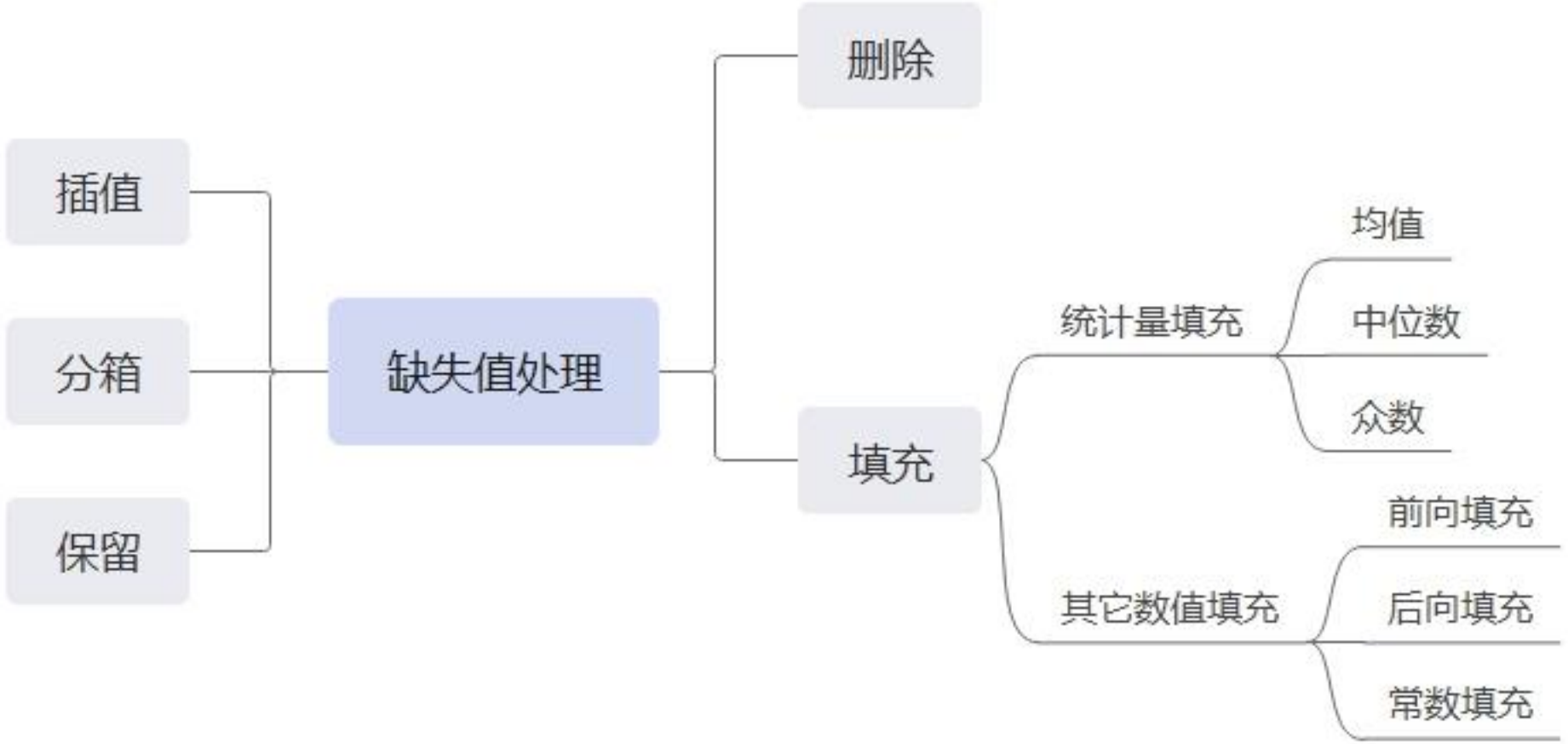
```
>>> dict4 = {'A':[1,2,np.NaN,4],'B':[3,4,4,5],'C':[5,6,7,8],'D':[7,5,np.NaN,np.NaN]}
>>> df4 = pd.DataFrame(dict4)
>>> print(df4)
     A  B  C    D
0  1.0  3  5  7.0
1  2.0  4  6  5.0
2  NaN  4  7  NaN
3  4.0  5  8  NaN
```

```
>>> 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——插值

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 1.5.0 documentation

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
```

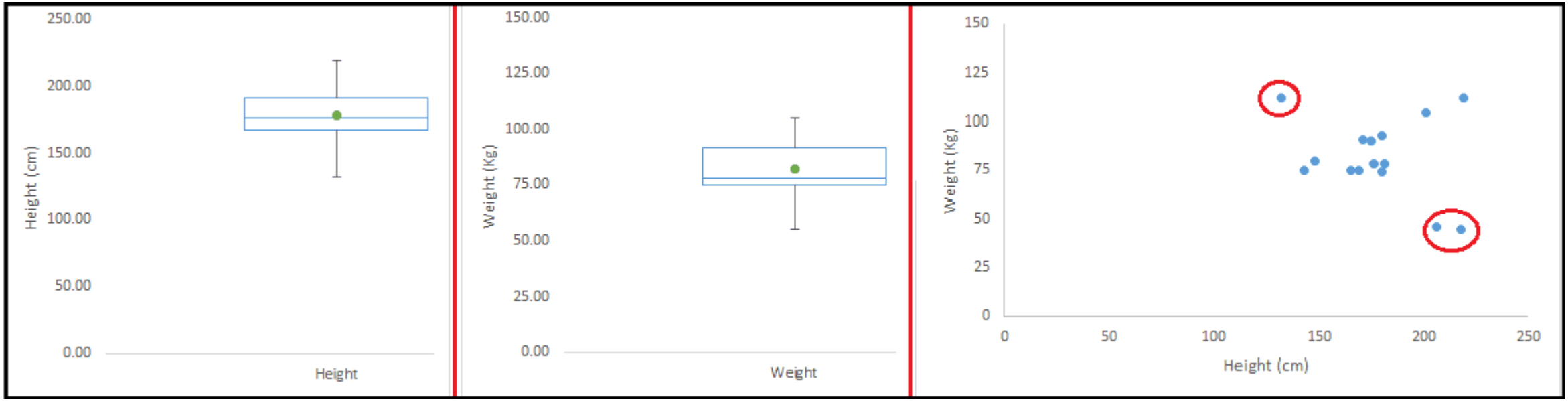
数据清理：重复值的删除

```
>>> 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

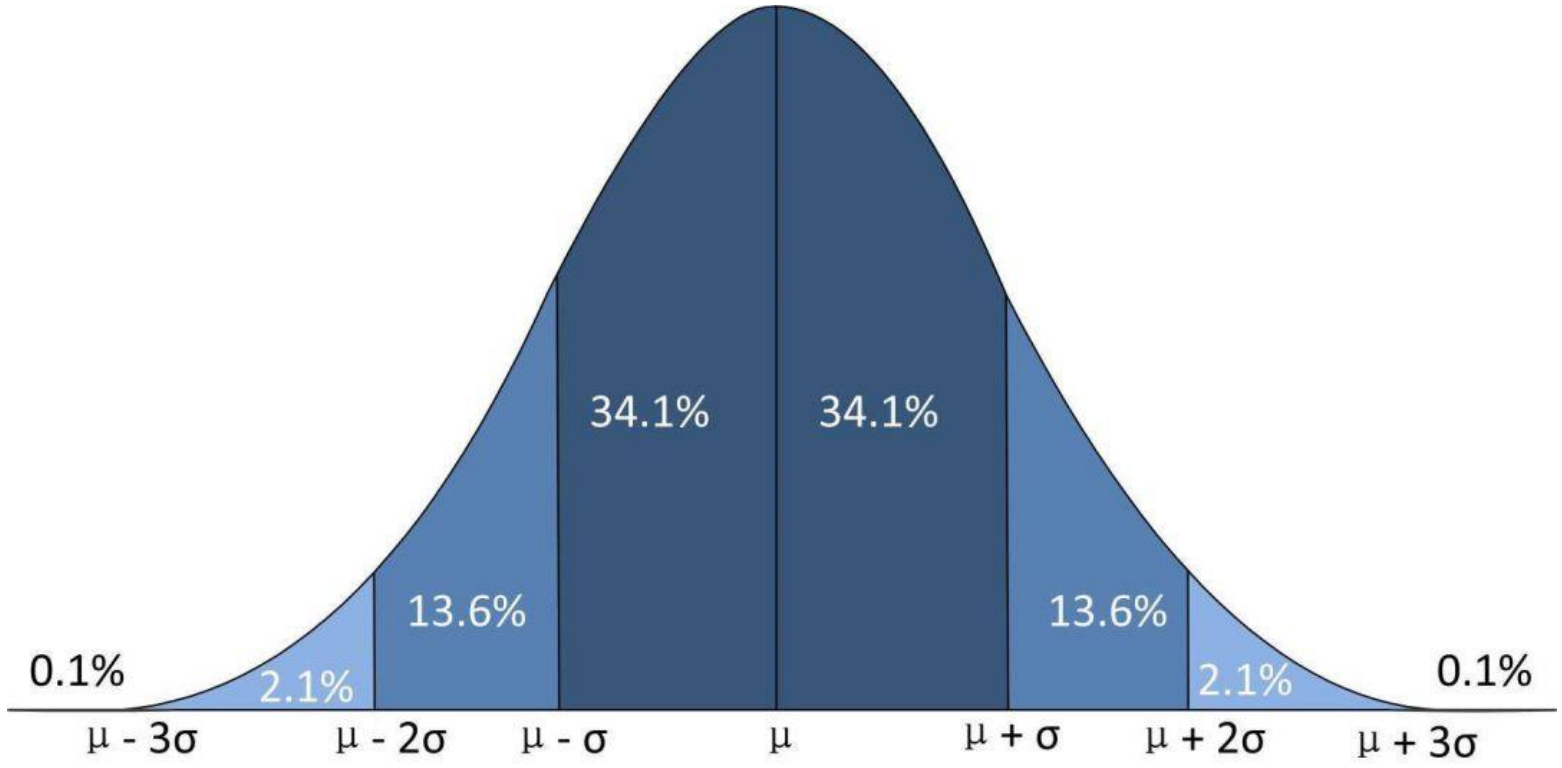


数据清理：异常值检测

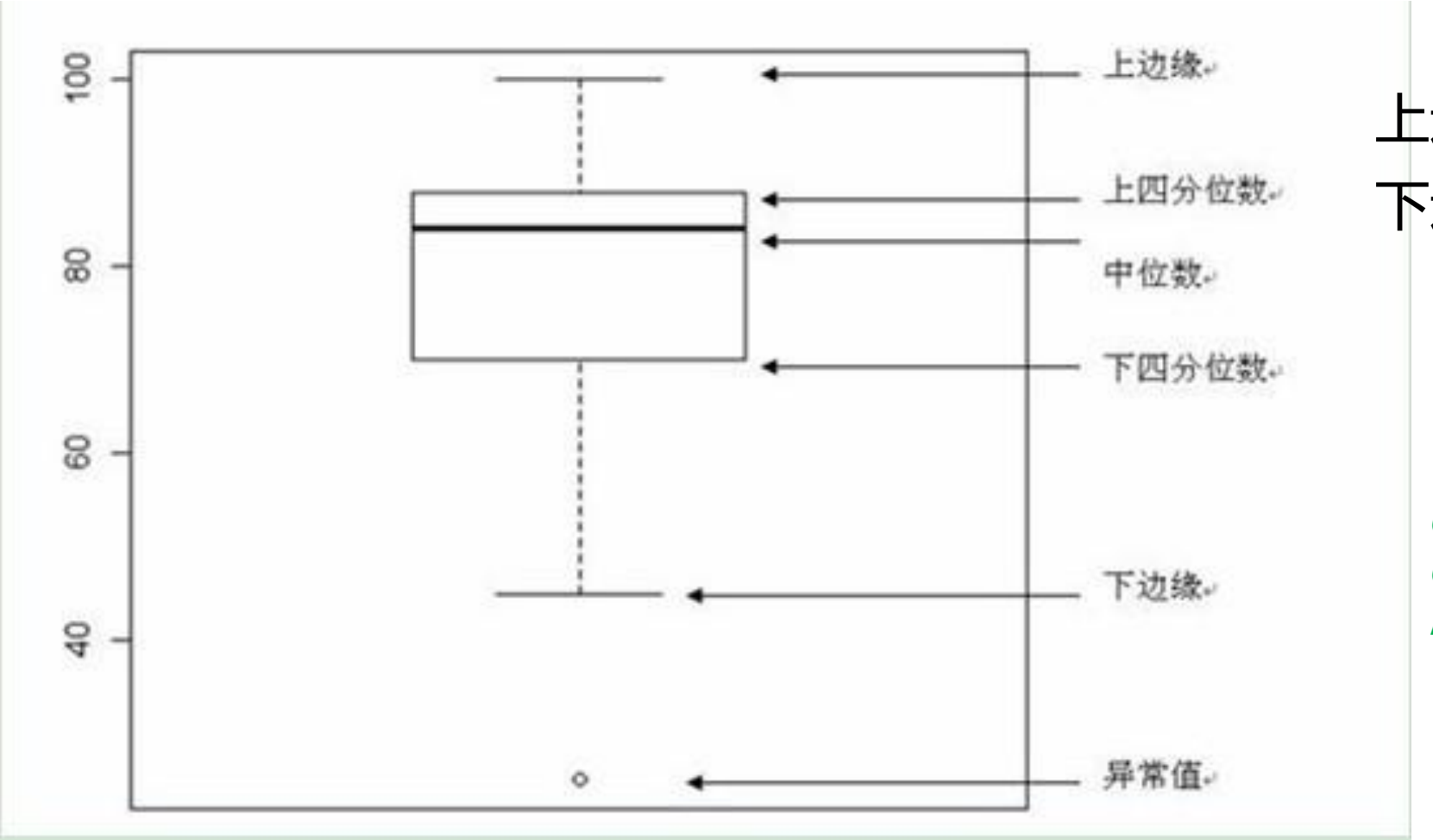
异常值的常用检测方法：

- 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

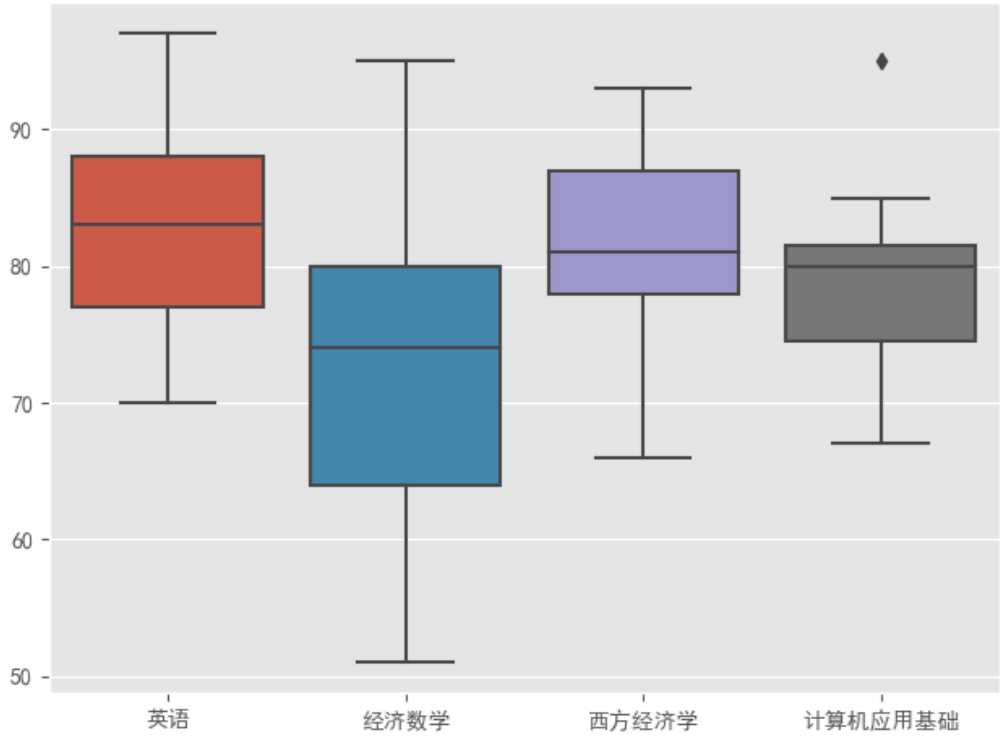


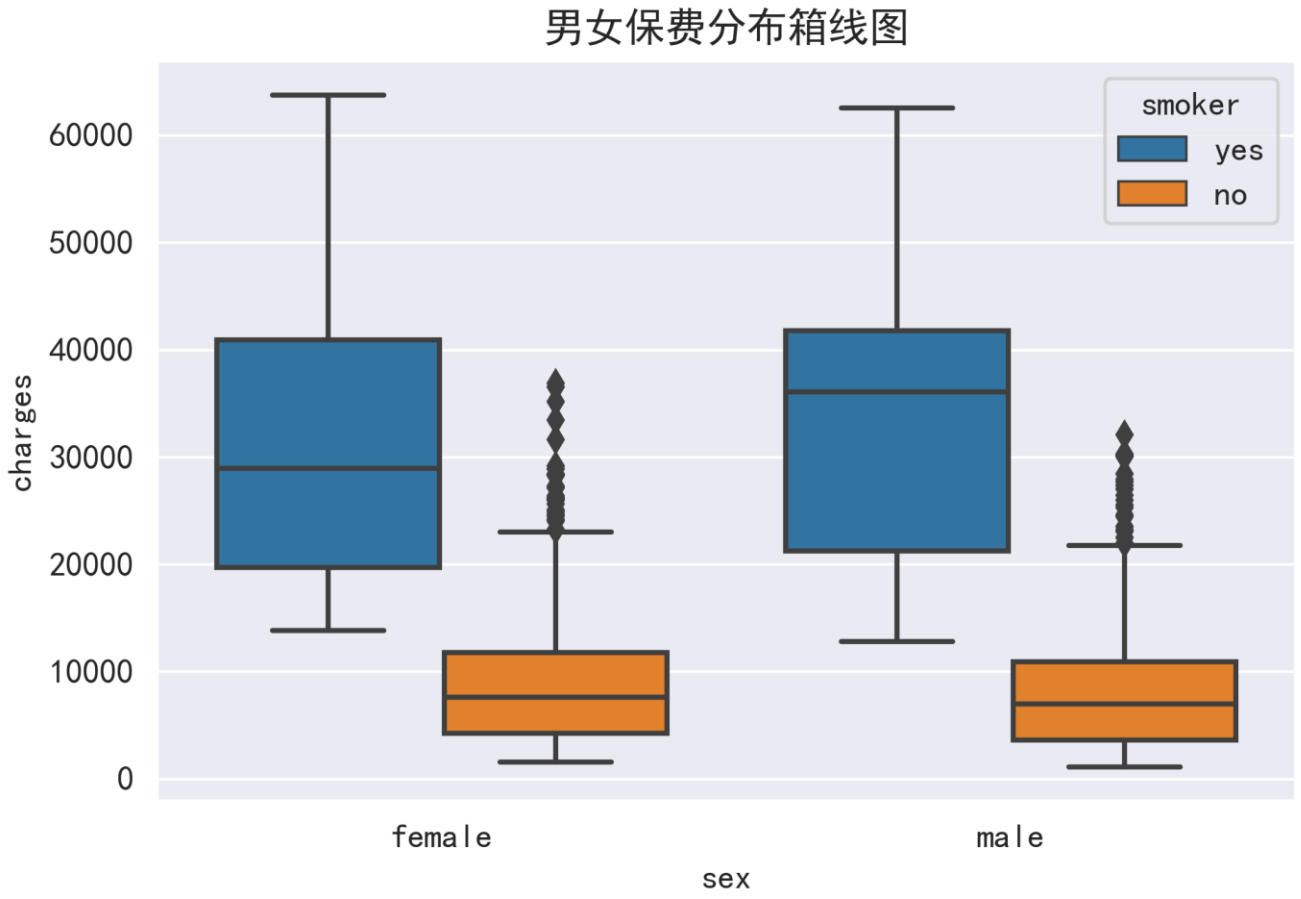
上边缘 =Q3 + kIQR

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





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)
```

- 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

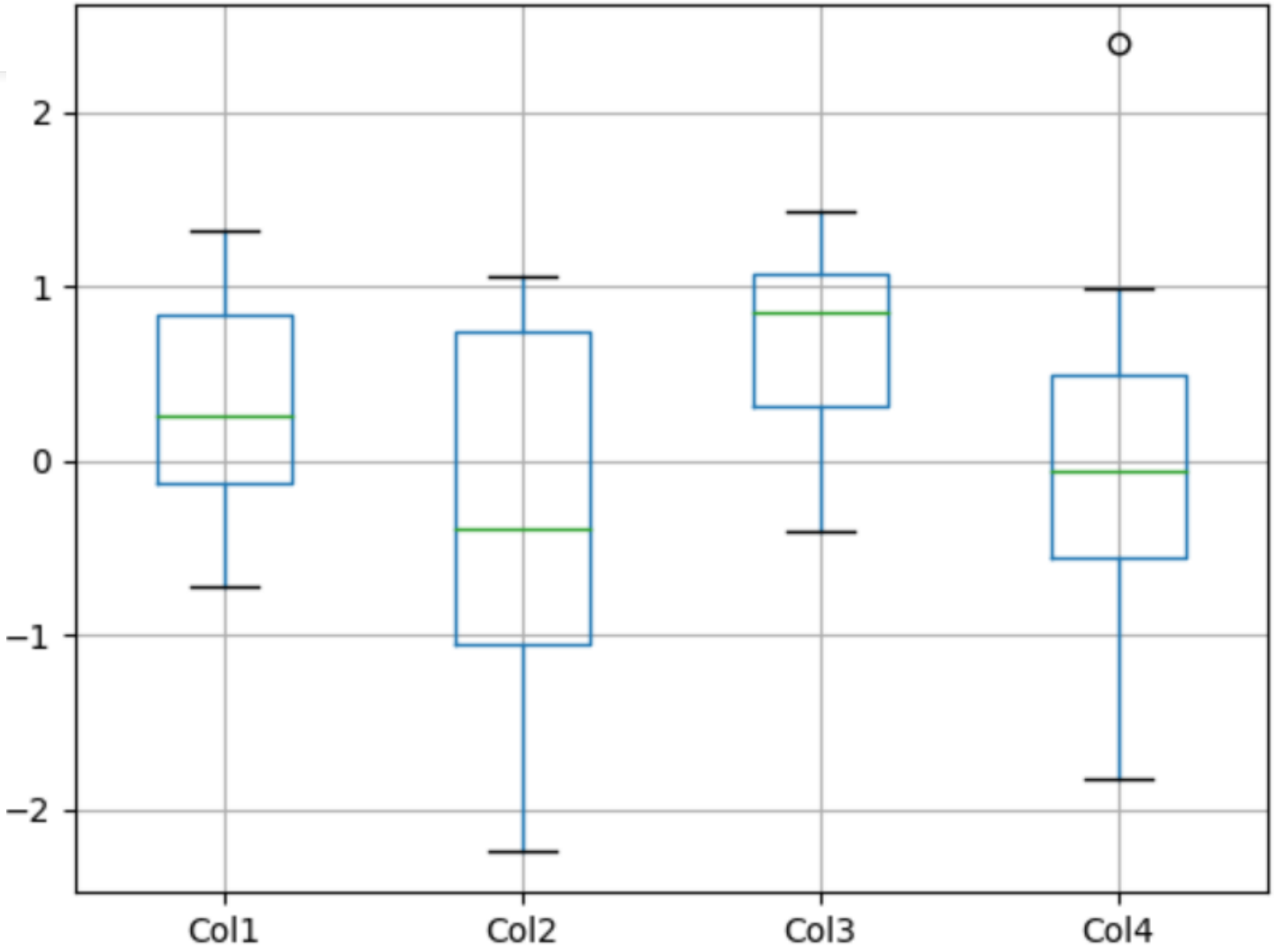
数据清理：异常值检测——箱形图

```
>>> 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
```

[pandas.DataFrame.boxplot — pandas 1.5.0 documentation]

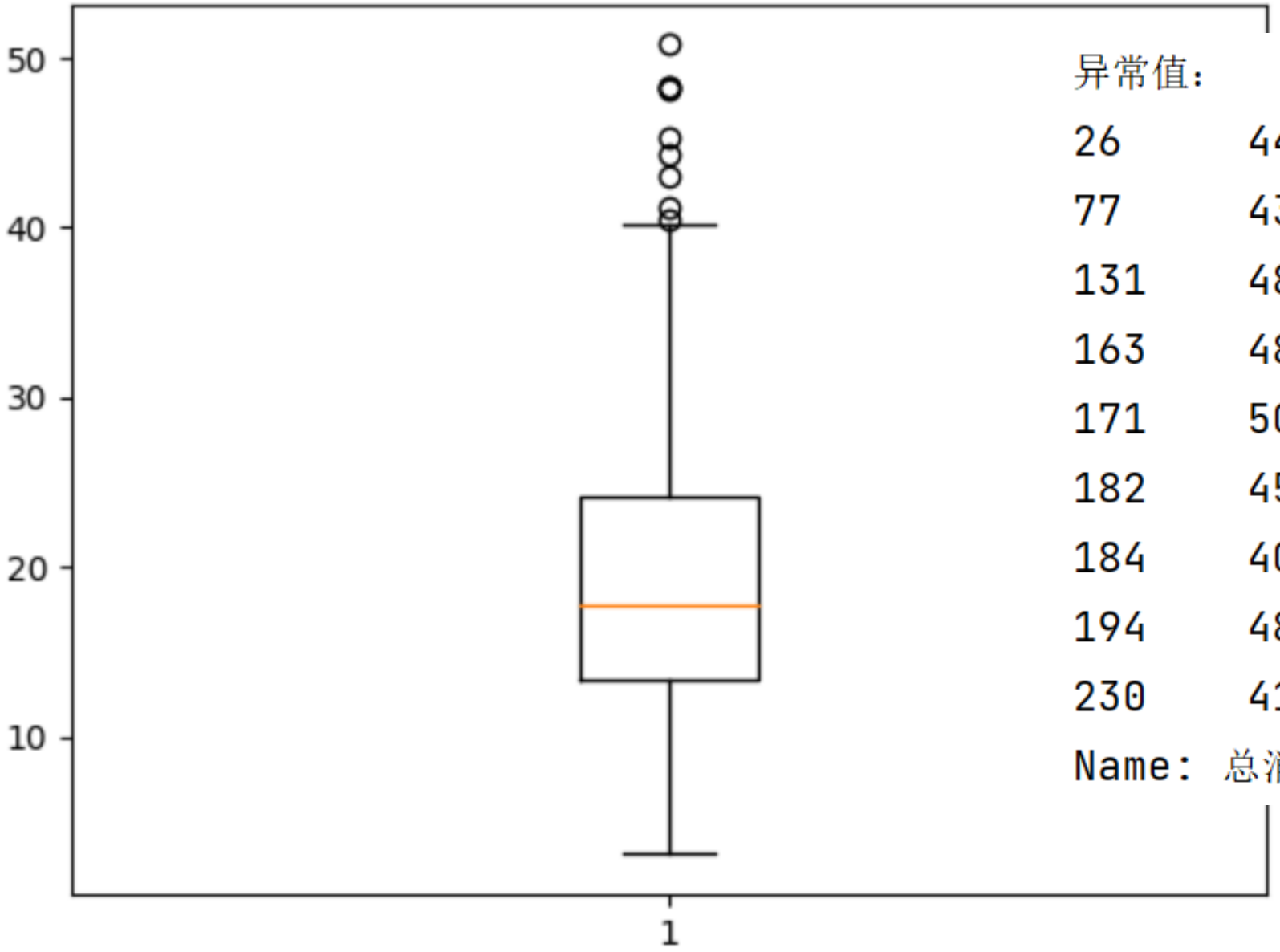
数据清理：异常值检测——箱形图

```
>>> 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)
```

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)
```

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         3
0  8.873036  2.164731  2.415485  0.781094
1  6.596223 -2.904616  9.234435  1.716379
2  4.957117  3.251889  8.386324 -2.180422
3  3.032748  2.847837  7.401308  0.700326
```

```
>>> 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

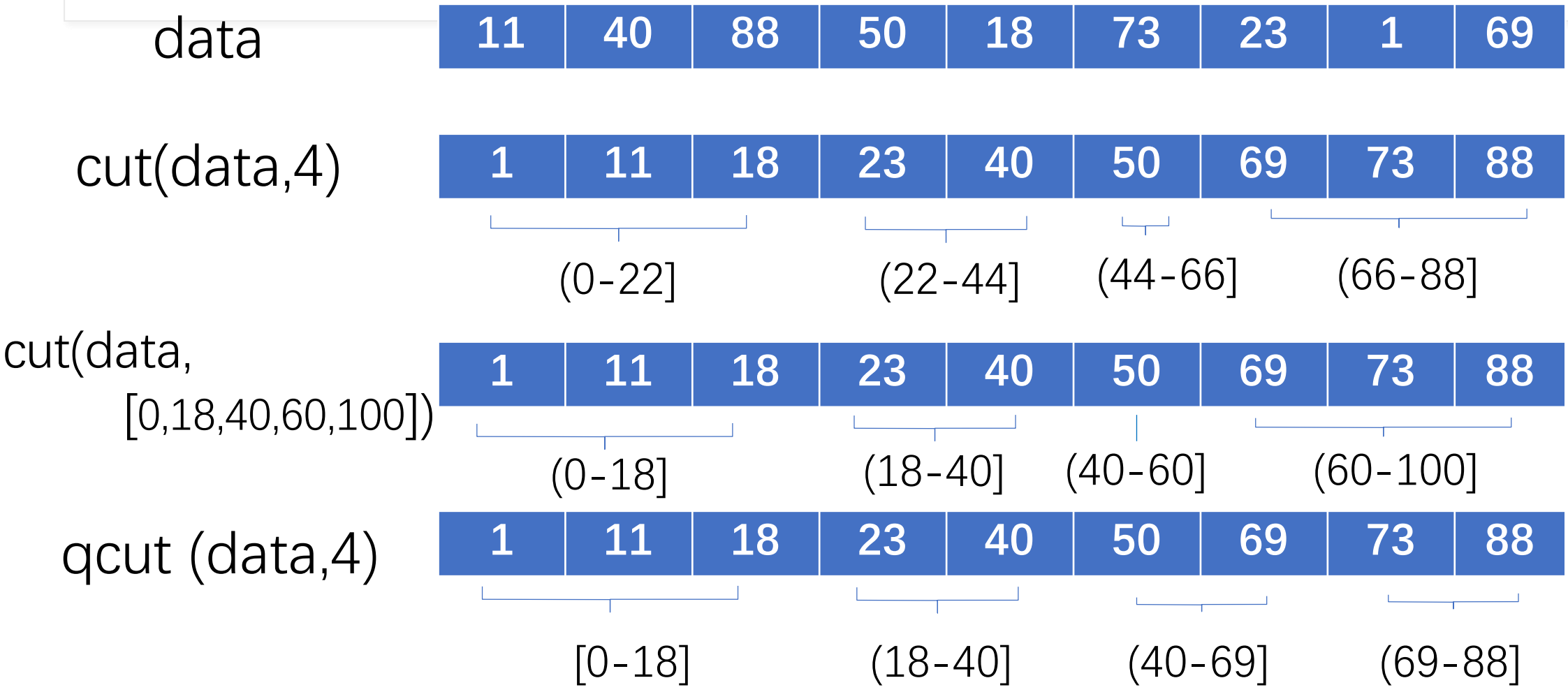
面元划分

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

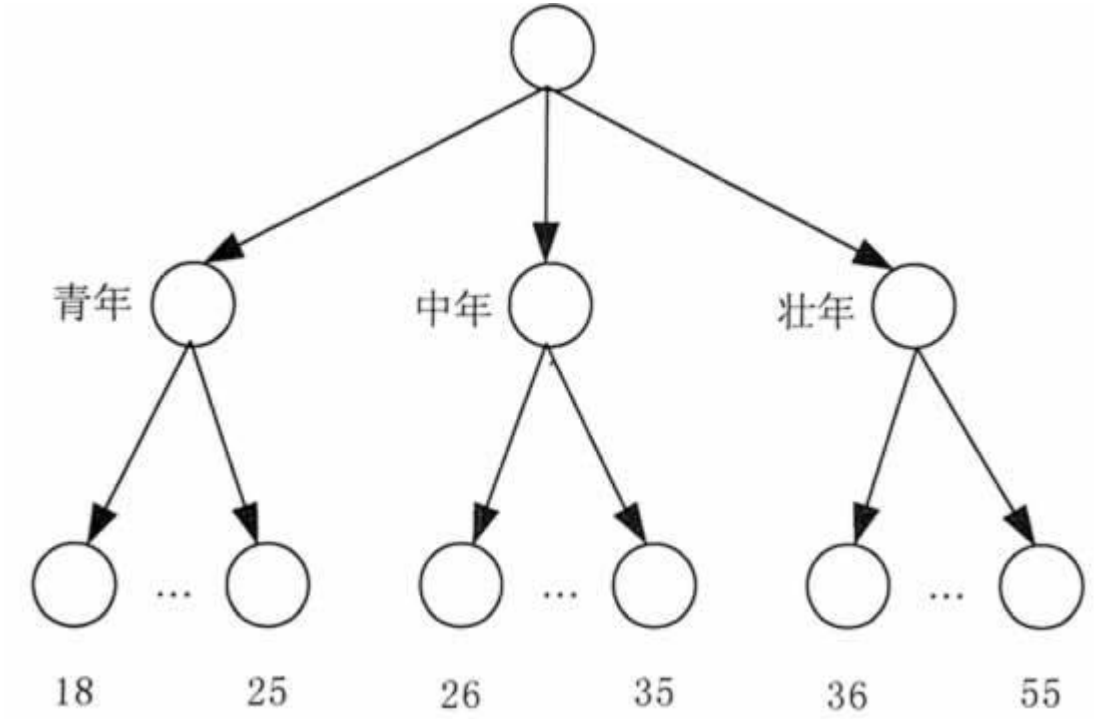
```
>>> cuts = pd.qcut(ages, q)
>>> cuts
0       (17.0, 29.0]
1       (17.0, 29.0]
2      (4.999, 17.0]
3       (42.0, 76.0]
4       (42.0, 76.0]
5      (4.999, 17.0]
6      (4.999, 17.0]
7       (29.0, 42.0]
8       (42.0, 76.0]
9       (29.0, 42.0]
10      (17.0, 29.0]
dtype: category
Categories (4,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

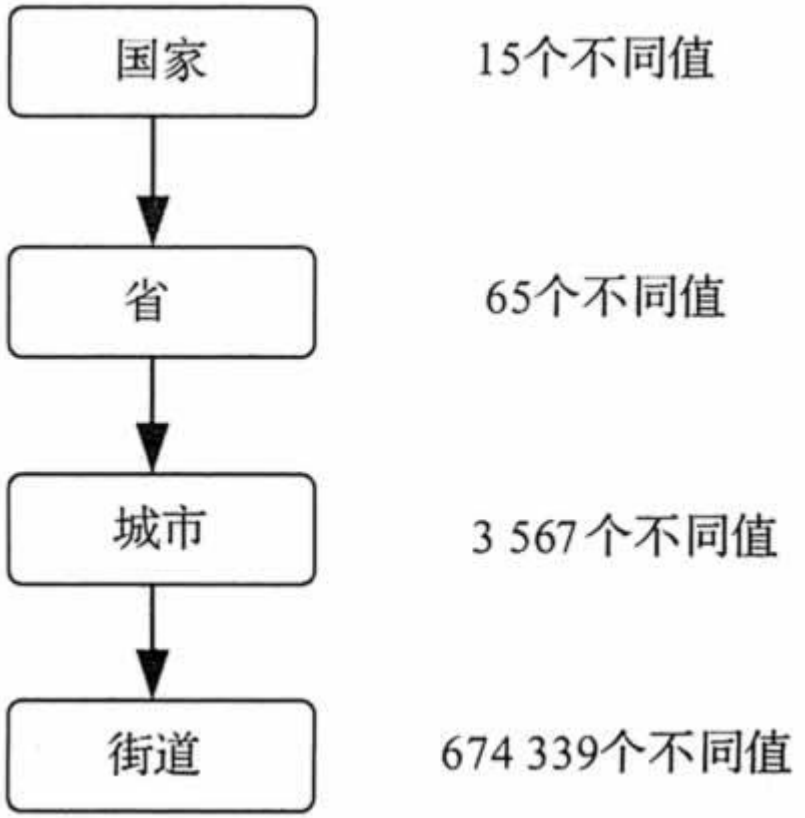
数据泛化

- 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
- 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]),
...                   columns=['A1','A2','A3','A4','A5'])
...
>>> print(df)
         A1        A2        A3        A4        A5
0  0.439211  0.003371  0.582835  0.173313  0.844386
1  0.477241  0.810613  0.667575  0.388158  0.385457
2  0.137104  0.272852  0.281588  0.021450  0.197956
3  0.362911  0.638299  0.128822  0.355853  0.236810
4  0.918478  0.304694  0.061689  0.851361  0.071521
```

数据格式化：设置小数位数

```
>>> print(df.round(2))
     A1    A2    A3    A4    A5
0  0.44  0.00  0.58  0.17  0.84
1  0.48  0.81  0.67  0.39  0.39
2  0.14  0.27  0.28  0.02  0.20
3  0.36  0.64  0.13  0.36  0.24
4  0.92  0.30  0.06  0.85  0.07
```

```
>>> print(df.round({'A1':1,'A2':2}))
    A1    A2        A3        A4        A5
0  0.4  0.00  0.582835  0.173313  0.844386
1  0.5  0.81  0.667575  0.388158  0.385457
2  0.1  0.27  0.281588  0.021450  0.197956
3  0.4  0.64  0.128822  0.355853  0.236810
4  0.9  0.30  0.061689  0.851361  0.071521
```

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_csv('Beijing_weather_202208.csv', encoding='GBK')
>>> print(df)
```

	日期	最高气温	最低气温	天气	风向
0	2022-08-01 星期一	34℃	25℃	多云	南风 1级
1	2022-08-02 星期二	34℃	26℃	多云	东北风 2级
2	2022-08-03 星期三	35℃	26℃	多云	东北风 2级
3	2022-08-04 星期四	35℃	26℃	多云	东南风 1级
4	2022-08-05 星期五	36℃	27℃	晴	西南风 2级
5	2022-08-06 星期六	33℃	25℃	雾	北风 1级
6	2022-08-07 星期日	32℃	25℃	雾	西北风 4级
7	2022-08-08 星期一	27℃	22℃	多云	南风 1级
8	2022-08-09 星期二	24℃	20℃	小雨	南风 2级

数据行列转换：一列数据转换为多列数据

Series.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

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

```
>>> print(df.head())
          日期 最高气温 最低气温   天气          风向     星期
0  2022-08-01     34℃     25℃   多云     南风 1级   星期一
1  2022-08-02     34℃     26℃   多云   东北风 2级   星期二
2  2022-08-03     35℃     26℃   多云   东北风 2级   星期三
3  2022-08-04     35℃     26℃   多云   东南风 1级   星期四
4  2022-08-05     36℃     27℃    晴   西南风 2级   星期五
```

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='"', lineterminator=None,
chunksize=None, date_format=None, doublequote=True, escapechar=None, decimal='.',
errors='strict', storage_options=None)
```

[source]

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=_NoDefault.no_default, inf_rep='inf', verbose=_NoDefault.no_default,
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