

Python 数据处理

刘德华

2023 年 7 月 21 日

1 基本技能

1.0.1 以 `openml` 中的数据集 `diabetes` 为例，计算不同 `class` 分类下，`age` 的中位数。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml

[2]: # 获取数据
data = fetch_openml(
    data_id=37,
    as_frame=True,
    parser="pandas"
)["frame"]

[3]: # 查看数据集的前五行
data.head()
```

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

```
[4]: # 分组计算平均数
data.groupby(by="class")["age"].agg("median")

[4]: class
tested_negative    27.0
tested_positive    36.0
Name: age, dtype: float64
```

1.0.2 以 `openml` 中的数据集 `diabetes` 为例，查找第五大的年龄是多少岁。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml

[2]: # 获取数据
data = fetch_openml(
    data_id=37,
    as_frame=True,
    parser="pandas"
)["frame"]

[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      preg  plas  pres  skin  insu  mass  pedi  age      class
0       6   148   72   35    0  33.6  0.627  50  tested_positive
1       1    85   66   29    0  26.6  0.351  31  tested_negative
2       8   183   64    0    0  23.3  0.672  32  tested_positive
3       1    89   66   23   94  28.1  0.167  21  tested_negative
4       0   137   40   35  168  43.1  2.288  33  tested_positive
```

```
[4]: # 降序排列后取出第四个
data["age"].sort_values(ascending=False).iloc[3]
```

```
[4]: 69
```

1.0.3 以 `openml` 中的数据集中 `diabetes` 为例，新建一个 `id` 列，划分为两个子数据集，两个数据集的样本量是一样的，只是变量不同。

1.0.4 将这两个的子数据集打乱顺序，按照 `id` 合并。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=37,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      preg  plas  pres  skin  insu  mass  pedi  age      class
0       6   148   72   35    0  33.6  0.627  50  tested_positive
1       1    85   66   29    0  26.6  0.351  31  tested_negative
2       8   183   64    0    0  23.3  0.672  32  tested_positive
3       1    89   66   23   94  28.1  0.167  21  tested_negative
4       0   137   40   35  168  43.1  2.288  33  tested_positive
```

```
[4]: # 新建一列 id
data["id"] = range(1,data.shape[0]+1)
```

```
[5]: # 按照 preg 排序，打乱顺序
data1 = data.loc[:, ["preg", "plas", "pres", "class", "id"]].
    ↪sort_values(by=["preg"])
# 按照 age 排序，打乱顺序
data2 = data.loc[:, ["skin", "insu", "mass", "pedi", "age", "id"]].
    ↪sort_values(by=["age"])
```

```
[6]: data1.head()
```

```
[6]:      preg  plas  pres      class  id
      467    0   97   64 tested_negative 468
      109    0   95   85 tested_positive 110
      452    0   91   68 tested_negative 453
      449    0  120   74 tested_negative 450
      448    0  104   64 tested_positive 449
```

```
[7]: data2.head()
```

```
[7]:      skin  insu  mass  pedi  age  id
      255   35    0  33.6  0.543  21  256
      60    0    0   0.0  0.304  21   61
      102    0    0  22.5  0.262  21  103
      182   20   23  27.7  0.299  21  183
      623   27  115  43.5  0.347  21  624
```

```
[8]: # 按照 id 合并
newdata = data1.merge(data2, on="id")
# 按照 id 排序
newdata.sort_values(by=["id"], inplace=True)
newdata.head()
```

```
[8]:      preg  plas  pres      class  id  skin  insu  mass  pedi  age
      587    6  148   72 tested_positive  1   35    0  33.6  0.627  50
      233    1   85   66 tested_negative  2   29    0  26.6  0.351  31
      665    8  183   64 tested_positive  3    0    0  23.3  0.672  32
      212    1   89   66 tested_negative  4   23   94  28.1  0.167  21
      101    0  137   40 tested_positive  5   35  168  43.1  2.288  33
```

1.0.5 以 openml 中的数据 1StudentPerformance 为例，求数学分数 math score 的排名

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      gender  race/ethnicity  parental level of education  lunch \
0  female      group B      bachelor's degree      standard
1  female      group C      some college      standard
```

2	female	group B	master\'s degree	standard
3	male	group A	associate\'s degree	free/reduced
4	male	group C	some college	standard

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

[4]: # 按照 *max* 方法进行排序

```
res1 = data["math score"].rank(method="max")
res1.sort_values().iloc[:12]
```

```
[4]: 59      1.0
      980      2.0
      17      3.0
      787      4.0
      145      5.0
      842      6.0
      338      7.0
      466      8.0
      91      10.0
      363      10.0
      327      11.0
      528      14.0
      Name: math score, dtype: float64
```

[5]: # 按照 *min* 方法进行排序

```
res2 = data["math score"].rank(method="min")
res2.sort_values().iloc[:12]
```

```
[5]: 59      1.0
      980      2.0
      17      3.0
      787      4.0
      145      5.0
      842      6.0
      338      7.0
      466      8.0
      91      9.0
      363      9.0
      327      11.0
      528      12.0
      Name: math score, dtype: float64
```

```
[6]: # 按照 dense 方法进行排序
res3 = data["math score"].rank(method="dense")
res3.sort_values().iloc[:12]
```

```
[6]: 59      1.0
     980     2.0
     17      3.0
     787     4.0
     145     5.0
     842     6.0
     338     7.0
     466     8.0
     91      9.0
     363     9.0
     327    10.0
     528    11.0
     Name: math score, dtype: float64
```

1.0.6 以 openml 中的数据集合 1StudentPerformance 为例，求数学分数 math score 连续出现 3 次的分数。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72         72         74
1         completed         69         90         88
2              none         90         95         93
```

3	none	47	57	44
4	none	76	78	75

```
[4]: # 一步差分结果, 删除缺失值
diff1 = data["math score"].diff().dropna()
diff1
```

```
[4]: 1      -3.0
     2      21.0
     3     -43.0
     4      29.0
     5      -5.0
     ...
    995      25.0
    996     -26.0
    997      -3.0
    998       9.0
    999       9.0
Name: math score, Length: 999, dtype: float64
```

```
[5]: # 找到为零的那些元素, 表示重复两次出现的值
diff1_zero = diff1[diff1==0]
diff1_zero
```

```
[5]: 384      0.0
     390      0.0
     432      0.0
     437      0.0
     453      0.0
     518      0.0
     537      0.0
     550      0.0
     565      0.0
     747      0.0
     797      0.0
     925      0.0
     948      0.0
Name: math score, dtype: float64
```

```
[6]: # 再对一步差分结果中零的索引进行差分, 若为零, 则说明是三个连续值
result = pd.Series(diff1_zero.index).diff().dropna()
result[result==0]
```

```
[6]: Series([], dtype: float64)
```

1.0.7 以 `openml` 中的数据集 `1StudentPerformance` 为例, 求阅读分数 `reading score` 中出现次数大于 1 的重复值。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none           72           72           74
1      completed           69           90           88
2              none           90           95           93
3              none           47           57           44
4              none           76           78           75
```

```
[8]: # 找出 reading score 值相同的行
dup_bool = data["reading score"].duplicated()
res = data["reading score"][dup_bool]
res
```

```
[8]: 6      95
    13     72
    21     75
    22     54
    26     54
    ..
   995     99
   996     55
   997     71
   998     78
```



```
999      86
Name: reading score, Length: 928, dtype: int64
```

1.0.8 以 `openml` 中的数据集中 `1StudentPerformance` 为例，求 `reading score` 中重复值各自出现的次数。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1        completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

```
[4]: # 频数统计
counts = data["reading score"].value_counts()
# 取出频数大于 1 的值
results = counts[counts > 1]
# 按照 reading score 的数值排序
res_sort = results.sort_index(ascending=False)
res_sort
```

```
[4]: 100      17
     99       3
     97       5
     96       4
```

```

95      8
..
37      3
34      4
31      2
29      2
24      2
Name: reading score, Length: 66, dtype: int64

```

1.0.9 以 `openml` 中的数据集 `1StudentPerformance` 为例，求 `math score` 中分数为零的样本。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

```

```

      test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75

```

```

[4]: # 数学分数为零的样本
data[data["math score"] == 0]

```

```

[4]:   gender race/ethnicity parental level of education      lunch \
59  female      group C      some high school  free/reduced

```

	test preparation course	math score	reading score	writing score
59	none	0	17	10

1.0.10 以 openml 中的数据集合 1StudentPerformance 为例，删除 writing score 中分数相同的样本。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard
```

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: newdata = data.drop_duplicates(subset=["writing score"])
print("删除 writing score 重复值后的样本量: ", newdata.shape, sep="\n")
newdata.head()
```

删除 writing score 重复值后的样本量:
(77, 8)

```
[4]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
```

4	male	group C	some college	standard
---	------	---------	--------------	----------

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

1.0.11 以 `openml` 中的数据集合 `1StudentPerformance` 为例，找到下一个 **math score** 分数比上一个高的样本值。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

[3]:	gender	race/ethnicity	parental level of education	lunch \
0	female	group B	bachelor\'s degree	standard
1	female	group C	some college	standard
2	female	group B	master\'s degree	standard
3	male	group A	associate\'s degree	free/reduced
4	male	group C	some college	standard

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 一步差分结果
diff_res = data["math score"].diff().dropna()
# 从一步差分结果的索引找到这些样本
data.loc[diff_res[diff_res > 0].index, ].head()
```

```
[4]:      gender race/ethnicity parental level of education      lunch \
2   female      group B      master\'s degree      standard
4    male      group C      some college      standard
6   female      group B      some college      standard
8    male      group D      high school free/reduced
10   male      group C      associate\'s degree      standard

      test preparation course  math score  reading score  writing score
2              none          90          95          93
4              none          76          78          75
6      completed          88          95          92
8      completed          64          64          67
10             none          58          54          52
```

1.0.12 以 openml 中的数据集合 1StudentPerformance 为例，找到 writing score 中分数大于 90 的样本。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      gender race/ethnicity parental level of education      lunch \
0   female      group B      bachelor\'s degree      standard
1   female      group C      some college      standard
2   female      group B      master\'s degree      standard
3    male      group A      associate\'s degree free/reduced
4    male      group C      some college      standard

      test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75
```

```
[4]: # 分数大于 90 的样本
data[data["writing score"] > 90].head()
```

```
[4]:      gender race/ethnicity parental level of education      lunch \
2    female      group B      master\'s degree      standard
6    female      group B      some college      standard
94   female      group B      some college      standard
106  female      group D      master\'s degree      standard
110  female      group D      associate\'s degree free/reduced

      test preparation course  math score  reading score  writing score
2              none          90          95          93
6      completed          88          95          92
94             none          79          86          92
106             none          87         100         100
110      completed          77          89          98
```

1.0.13 以 openml 中的数据集 1StudentPerfromance 为例，统计各个民族的人数。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      gender race/ethnicity parental level of education      lunch \
0    female      group B      bachelor\'s degree      standard
1    female      group C      some college      standard
2    female      group B      master\'s degree      standard
3     male      group A      associate\'s degree free/reduced
4     male      group C      some college      standard

      test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
```

4	none	76	78	75
---	------	----	----	----

```
[4]: # 分组统计频数
data["race/ethnicity"].value_counts()
```

```
[4]: group C    319
      group D    262
      group B    190
      group E    140
      group A     89
      Name: race/ethnicity, dtype: int64
```

1.0.14 以 `openml` 中的数据集合 `1StudentPerformance` 为例，寻找 `math score` 中只出现一次的最大数值。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education    lunch \
0  female      group B      bachelor's degree    standard
1  female      group C              some college    standard
2  female      group B      master's degree    standard
3   male      group A      associate's degree  free/reduced
4   male      group C              some college    standard
```

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 寻找唯一值，再从唯一值中求最大值
data["math score"].unique().max()
```

```
[4]: 100
```

1.0.15 以 `openml` 中的数据集合 `1StudentPerfromance` 为例，将 `reading score` 中数值为 `59` 的全部修改为 `60`。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none           72           72           74
1        completed           69           90           88
2              none           90           95           93
3              none           47           57           44
4              none           76           78           75
```

```
[4]: # 将 reading score 为 59 的替换为 60
print("替换之前，数字 60 的个数为：", data["reading score"].value_counts().loc[60,])
data["reading score"].replace(to_replace=59, value=60, inplace=True)
print("替换之后，数字 60 的个数为：", data["reading score"].value_counts().loc[60,])
```

替换之前，数字 60 的个数为： 21

替换之后，数字 60 的个数为： 38

1.0.16 以 `openml` 中的数据集合 `CSM` 为例，计算评分 `Ratings` 前 10 的电影到底有多少人喜欢？

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```



```
[2]: # 获取数据
data = fetch_openml(
    data_id=42371,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:
```

	Year	Ratings	Genre	Gross	Budget	Screens	Sequel	Sentiment	\
0	2014	6.3	8	9130	4000000.0	45.0	1	0	
1	2014	7.1	1	192000000	50000000.0	3306.0	2	2	
2	2014	6.2	1	30700000	28000000.0	2872.0	1	0	
3	2014	6.3	1	106000000	110000000.0	3470.0	2	0	
4	2014	4.7	8	17300000	3500000.0	2310.0	2	0	

	Views	Likes	Dislikes	Comments	Aggregate.Followers
0	3280543	4632	425	636	1120000.0
1	583289	3465	61	186	12350000.0
2	304861	328	34	47	483000.0
3	452917	2429	132	590	568000.0
4	3145573	12163	610	1082	1923800.0

```
[4]: # 按照 Rating 降序排列，取出前 10 对应的 Likes 数
data.sort_values(by=["Ratings"], ascending=False).iloc[:10,][["Ratings", "Likes"]]
```

```
[4]:
```

	Ratings	Likes
55	8.7	16635
166	8.6	4632
155	8.6	17541
174	8.3	13030
175	8.3	12607
158	8.2	1023
45	8.2	1390
212	8.2	18398
46	8.1	8567
129	8.1	11748

1.0.17 以 `openml` 中的数据集 **CSM** 为例，将数据集按照评分 **Ratings** 作为第一因子降序排列，**Likes** 作为第二因子升序排列。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=42371,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      Year  Ratings  Genre      Gross      Budget  Screens  Sequel  Sentiment  \
0   2014      6.3      8      9130    4000000.0      45.0      1          0
1   2014      7.1      1  192000000    50000000.0     3306.0      2          2
2   2014      6.2      1   30700000    28000000.0     2872.0      1          0
3   2014      6.3      1  106000000   110000000.0     3470.0      2          0
4   2014      4.7      8   17300000    3500000.0     2310.0      2          0
```

```
      Views  Likes  Dislikes  Comments  Aggregate.Followers
0   3280543   4632      425      636          1120000.0
1    583289   3465       61      186          12350000.0
2    304861    328       34       47           483000.0
3    452917   2429      132      590           568000.0
4   3145573  12163      610     1082          1923800.0
```

```
[4]: # 多字段排序，升降序排列
res = data.sort_values(by=["Ratings", "Likes"], ascending=[False, True])
res.head()
```

```
[4]:      Year  Ratings  Genre      Gross      Budget  Screens  Sequel  Sentiment  \
55   2014      8.7      2  188000000   165000000.0     3561.0      1          2
166  2015      8.6     12  345000000   175000000.0     3946.0      1          2
155  2014      8.6      3   13100000    3300000.0      42.0      1          2
175  2015      8.3      9  135000000    28000000.0     2757.0      1          5
174  2015      8.3      1  153000000   150000000.0     3702.0      4         -4
```

```
      Views  Likes  Dislikes  Comments  Aggregate.Followers
55   5421705  16635      751     4316          1865000.0
166  1438926   4632      262      496           232000.0
155  7750223  17541      631     2760           858000.0
175   848970  12607      237     1560           55618.0
174  2732371  13030      497     1774           768700.0
```

1.0.18 以 `openml` 中的数据集合 **CSM** 为例，将数据集的行索引修改为 **Movie{i}**。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=42371,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      Year  Ratings  Genre      Gross      Budget  Screens  Sequel  Sentiment  \
0  2014      6.3      8      9130  4000000.0      45.0      1          0
1  2014      7.1      1  192000000  50000000.0     3306.0      2          2
2  2014      6.2      1   30700000  28000000.0     2872.0      1          0
3  2014      6.3      1  106000000  110000000.0     3470.0      2          0
4  2014      4.7      8   17300000   3500000.0     2310.0      2          0

      Views  Likes  Dislikes  Comments  Aggregate.Followers
0  3280543  4632      425      636      1120000.0
1   583289  3465      61      186     12350000.0
2   304861   328      34      47      483000.0
3   452917  2429     132     590     568000.0
4  3145573 12163     610    1082    1923800.0
```

```
[4]: # 重新设置行索引
data.index = ["Movie{}".format(i) for i in range(1, data.shape[0]+1)]
data.head()
```

```
[4]:      Year  Ratings  Genre      Gross      Budget  Screens  Sequel  \
Movie1  2014      6.3      8      9130  4000000.0      45.0      1
Movie2  2014      7.1      1  192000000  50000000.0     3306.0      2
Movie3  2014      6.2      1   30700000  28000000.0     2872.0      1
Movie4  2014      6.3      1  106000000  110000000.0     3470.0      2
Movie5  2014      4.7      8   17300000   3500000.0     2310.0      2

      Sentiment  Views  Likes  Dislikes  Comments  Aggregate.Followers
Movie1          0  3280543  4632      425      636      1120000.0
Movie2          2   583289  3465      61      186     12350000.0
Movie3          0   304861   328      34      47      483000.0
Movie4          0   452917  2429     132     590     568000.0
```

Movie5	0	3145573	12163	610	1082	1923800.0
--------	---	---------	-------	-----	------	-----------

1.0.19 以 openml 中的数据集合 CSM 为例，将数据集的部分列名进行修改。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=42371,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:
```

	Year	Ratings	Genre	Gross	Budget	Screens	Sequel	Sentiment	\
0	2014	6.3	8	9130	4000000.0	45.0	1	0	
1	2014	7.1	1	192000000	50000000.0	3306.0	2	2	
2	2014	6.2	1	30700000	28000000.0	2872.0	1	0	
3	2014	6.3	1	106000000	110000000.0	3470.0	2	0	
4	2014	4.7	8	17300000	3500000.0	2310.0	2	0	

	Views	Likes	Dislikes	Comments	Aggregate.Followers
0	3280543	4632	425	636	1120000.0
1	583289	3465	61	186	12350000.0
2	304861	328	34	47	483000.0
3	452917	2429	132	590	568000.0
4	3145573	12163	610	1082	1923800.0

```
[4]: # 列重新命名
data.columns = [
    "年份",
    "评分",
    "类型",
    "gross",
    "预算",
    "screens",
    "续集",
    "感情",
    "观看次数",
    "喜欢人数",
    "不喜欢人数",
```

```

    "评论数",
    "粉丝数"
]
data.head()

```

```

[4]:      年份  评分  类型      gross      预算  screens  续集  感情  观看次数  \
      喜欢人数  \
0  2014  6.3   8      9130    4000000.0    45.0   1   0  3280543  4632
1  2014  7.1   1  192000000  50000000.0   3306.0   2   2   583289  3465
2  2014  6.2   1   30700000  28000000.0   2872.0   1   0   304861   328
3  2014  6.3   1  106000000  110000000.0   3470.0   2   0   452917  2429
4  2014  4.7   8   17300000   3500000.0   2310.0   2   0  3145573  12163

      不喜欢人数  评论数      粉丝数
0      425     636   1120000.0
1       61     186  12350000.0
2       34      47   483000.0
3      132     590   568000.0
4      610    1082  1923800.0

```

1.0.20 以 `openml` 中的数据集 CSM 为例，从数据集中无放回地随机抽取 50 个样本，计算评分 **Ratings** 的平均值。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=42371,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:      Year  Ratings  Genre      Gross      Budget  Screens  Sequel  Sentiment  \
0  2014      6.3     8      9130    4000000.0    45.0      1      0
1  2014      7.1     1  192000000  50000000.0   3306.0      2      2
2  2014      6.2     1   30700000  28000000.0   2872.0      1      0
3  2014      6.3     1  106000000  110000000.0   3470.0      2      0
4  2014      4.7     8   17300000   3500000.0   2310.0      2      0

      Views  Likes  Dislikes  Comments  Aggregate.Followers

```

0	3280543	4632	425	636	1120000.0
1	583289	3465	61	186	12350000.0
2	304861	328	34	47	483000.0
3	452917	2429	132	590	568000.0
4	3145573	12163	610	1082	1923800.0

```
[4]: # 随机抽样平均值
data["Ratings"].sample(n=50, replace=False).mean()
```

```
[4]: 6.534
```

1.0.21 以 openml 中的数据集 1StudentPerformance 为例，计算 math score 大于 60 且小于 90 的人数。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard
```

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 查找满足条件的样本
res = data[(data["math score"] > 60) & (data["math score"] < 90)]
# 计算样本量
res.shape[0]
```

[4]: 603

1.0.22 以 openml 中的数据集中 StudentPerformance 为例,找到 math score 小于 reading score 小于 writing score 的人。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

```
[4]: # 修改列名, 为了使用 query 语法
data.columns = data.columns[:5].to_list() + ["math_score", "reading_score", "writing_score"]
# 该查询方法直接是列名表达式
newdata = data.query("math_score < reading_score < writing_score")
newdata.head()
```

```
[4]:   gender race/ethnicity parental level of education      lunch \
14  female      group A      master\'s degree      standard
15  female      group C      some high school      standard
19  female      group C      associate\'s degree  free/reduced
27  female      group C      bachelor\'s degree      standard
29  female      group D      master\'s degree      standard
```

	test preparation course	math_score	reading_score	writing_score
14	none	50	53	58
15	none	69	75	78
19	none	54	58	61
27	none	67	69	75
29	none	62	70	75

1.0.23 以 openml 中的数据集中 StudentPerformance 为例, 查找 math score 中是否存在零分。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor's degree      standard
1  female      group C      some college      standard
2  female      group B      master's degree      standard
3   male      group A      associate's degree  free/reduced
4   male      group C      some college      standard
```

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: 0 in data["math score"]
```

```
[4]: True
```


1.0.24 以 `openml` 中的数据集 `1StudentPerformance` 为例, 查找 `math score` 的数值属于 `reading score` 的样本。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

   test preparation course  math score  reading score  writing score
0                none         72         72         74
1          completed         69         90         88
2                none         90         95         93
3                none         47         57         44
4                none         76         78         75
```

```
[4]: # 修改列名, 为了使用 query 语法
data.columns = data.columns[:5].to_list() + ["math", "reading", "writing"]
res = data.query("math in reading")
res.head()
```

```
[4]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

   test preparation course  math  reading  writing
0                none      72      72      74
1          completed      69      90      88
```

2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

1.0.25 以 openml 中的数据集中 StudentPerformance 为例, 查找 race/ethnicity 为 groupA 和 groupB 的人。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor's degree      standard
1  female      group C      some college      standard
2  female      group B      master's degree      standard
3   male      group A      associate's degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0             none           72           72           74
1      completed           69           90           88
2             none           90           95           93
3             none           47           57           44
4             none           76           78           75
```

```
[4]: # 修改列名
data["race"] = data["race/ethnicity"].copy()
# 删除原来的列
data.drop(columns=["race/ethnicity"], inplace=True)
res = data.query("race in ['group A', 'group C']")
res.head()
```

```
[4]:  gender parental level of education      lunch test preparation course \
1  female      some college      standard      completed
3   male      associate's degree  free/reduced      none
```

```

4      male      some college      standard      none
10     male      associate\'s degree      standard      none
13     male      some college      standard      completed

```

```

      math score  reading score  writing score  race
1           69           90           88 group C
3           47           57           44 group A
4           76           78           75 group C
10          58           54           52 group C
13          78           72           70 group A

```

1.0.26 以 `openml` 中的数据集合 `1StudentPerformance` 为例，将列名 `math score` 修改为 `MathScore`，将 `reading score` 修改为 `ReadingScore`，将 `writing score` 修改为 `WritingScore`。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[ ]: # 查看数据集的前五行
data.head()

```

```

[ ]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

```

```

      test preparation course  math score  reading score  writing score
0           none           72           72           74
1      completed           69           90           88
2           none           90           95           93
3           none           47           57           44
4           none           76           78           75

```

```

[ ]: # 修改列名
data.rename(columns={
    "math score": "MathScore",

```

```

    "reading score": "ReadingScore",
    "writing score": "WritingScore",
}, inplace=True)
data.head()

```

1.0.27 以 `openml` 中的数据集合 `1StudentPerformance` 为例，将数据集随机地拆分为三块，再从三个子集中分别进行重抽样，将所得到的结果按行合并。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72          72          74
1      completed         69          90          88
2              none         90          95          93
3              none         47          57          44
4              none         76          78          75

```

```

[4]: # 数据集划分
from sklearn.model_selection import train_test_split
# 第一次划分
df_train, df_test = train_test_split(data, test_size=0.3, random_state=1)
# 第二次划分
df_train1, df_test1 = train_test_split(df_train, test_size=0.3, random_state=2)
# 重抽样
sample1 = df_train1.sample(n=100, replace=True)
sample2 = df_test1.sample(n=100, replace=True)

```

```
sample3 = df_test.sample(n=100, replace=True)
# 按行合并
newdata = pd.concat([sample1, sample2, sample3], axis=0)
newdata.head()
```

```
[4]:      gender race/ethnicity parental level of education      lunch \
497  female      group D      some college free/reduced
384  female      group A      some high school free/reduced
487  female      group C  associate\'s degree free/reduced
273  female      group D      some college      standard
728  female      group D      high school free/reduced

      test preparation course  math score  reading score  writing score
497      completed          59          78          76
384      none              38          43          43
487      none              60          75          74
273      none              65          70          71
728      none              73          92          84
```

1.0.28 以 openml 中的数据集中 **1StudentPerformance** 为例，将数据集进行三次重抽样，将所得到的结果按行合并（每次重抽样的结果可能有样本值是相同的，合并时需要特别注意，我们这里重新设置行索引）。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:      gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A  associate\'s degree free/reduced
4   male      group C      some college      standard

      test preparation course  math score  reading score  writing score
```

0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[8]: # 重抽样
sample1 = data.sample(n=100, replace=True, random_state=1)
sample2 = data.sample(n=100, replace=True, random_state=2)
sample3 = data.sample(n=100, replace=True, random_state=3)
# 按行合并, 忽略行索引, 重新给定索引
newdata = pd.concat([sample1, sample2, sample3], axis=0, ignore_index=True)
newdata.head()
```

```
[8]:      gender race/ethnicity parental level of education      lunch \
0  female      group D      some high school  free/reduced
1   male      group D  associate\'s degree    standard
2  female      group C  bachelor\'s degree  free/reduced
3  female      group A  associate\'s degree  free/reduced
4   male      group B      high school    standard

      test preparation course  math score  reading score  writing score
0          none             50           64           59
1          none             80           75           77
2          none             67           75           72
3          none             41           51           48
4  completed             76           62           60
```

1.0.29 以 `openml` 中的数据集中 `1StudentPerformance` 为例, 将数据集拆分成两个子集 (变量拆分), 样本是从原数据集中重抽样所得, 将这两个子集按列合并, 默认键是行索引 `index`。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]: gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0          none          72          72          74
1      completed          69          90          88
2          none          90          95          93
3          none          47          57          44
4          none          76          78          75
```

```
[4]: # 重抽样
data1 = data.iloc[:, :4].sample(n=50, replace=True, random_state=1)
data2 = data.iloc[:, 4:].sample(n=50, replace=True, random_state=2)
# 按列合并，默认的 key 是 Index，取并集，以缺失值填充
newdata1 = pd.concat([data1, data2], axis=1, join="outer")
newdata1.head()
```

```
[4]: gender race/ethnicity parental level of education      lunch \
37  female      group D      some high school  free/reduced
235  male      group D      associate\'s degree      standard
908  female      group C      bachelor\'s degree  free/reduced
72   female      group A      associate\'s degree  free/reduced
767  male      group B      high school      standard

test preparation course  math score  reading score  writing score
37          NaN          NaN          NaN          NaN
235          NaN          NaN          NaN          NaN
908          NaN          NaN          NaN          NaN
72          NaN          NaN          NaN          NaN
767          NaN          NaN          NaN          NaN
```

```
[5]: # 按列合并，默认的 key 是 Index，取交集，无缺失值
newdata2 = pd.concat([data1, data2], axis=1, join="inner")
newdata2.head()
```

```
[5]: gender race/ethnicity parental level of education      lunch \
534  male      group B      high school      standard

test preparation course  math score  reading score  writing score
534      completed          73          69          68
```

1.0.30 以 `openml` 中的数据集 `1StudentPerformance` 为例，将列 `gender` 取出来转为 `dataframe` 对象。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

   test preparation course  math score  reading score  writing score
0             none         72             72             74
1      completed         69             90             88
2             none         90             95             93
3             none         47             57             44
4             none         76             78             75
```

```
[4]: # Series 转 Dataframe
data["gender"].to_frame()
```

```
[4]:   gender
0  female
1  female
2  female
3   male
4   male
..    ...
995 female
996  male
997 female
998 female
999 female
```


[1000 rows x 1 columns]

1.0.31 以 openml 中的数据集合 1StudentPerformance 为例，统计不同 race 和不同性别下数学成绩的平均值。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0          none          72          72          74
1      completed          69          90          88
2          none          90          95          93
3          none          47          57          44
4          none          76          78          75
```

```
[4]: # 在不同的分组下，计算数学成绩平均值
data.groupby(by=["gender", "race/ethnicity"])["math score"].agg("mean")
```

```
[4]: gender  race/ethnicity
female  group A      58.527778
        group B      61.403846
        group C      62.033333
        group D      65.248062
        group E      70.811594
male    group A      63.735849
        group B      65.930233
        group C      67.611511
```

```

        group D          69.413534
        group E          76.746479
Name: math score, dtype: float64

```

1.0.32 以 `openml` 中的数据集 `1StudentPerformance` 为例, 统计不同 `race` 和不同性别下阅读成绩的中位数。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor's degree      standard
1  female      group C              some college      standard
2  female      group B      master's degree      standard
3   male      group A      associate's degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75

```

```

[4]: # 数据透视表
pd.pivot_table(
    data,
    values="reading score",
    index="gender",
    columns="race/ethnicity",
    aggfunc="median"
)

```

```
[4]: race/ethnicity  group A  group B  group C  group D  group E
gender
female           67.5    71.5    73.0    74.0    76.0
male            61.0    62.0    66.0    68.0    73.0
```

1.0.33 以 openml 中的数据集 1StudentPerformance 为例，统计不同 race 和不同性别，不同的 parental level of education 下，写作成绩的方差。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education  lunch \
0  female      group B      bachelor's degree  standard
1  female      group C           some college  standard
2  female      group B      master's degree  standard
3   male      group A  associate's degree  free/reduced
4   male      group C           some college  standard

   test preparation course  math score  reading score  writing score
0             none         72           72           74
1        completed         69           90           88
2             none         90           95           93
3             none         47           57           44
4             none         76           78           75
```

```
[4]: # 数据透视表
pd.pivot_table(
    data,
    values="writing score",
    index=["gender", "parental level of education"],
    columns="race/ethnicity",
    aggfunc="var"
)
```

```
[4]: race/ethnicity          group A      group B      group C \
gender parental level of education
female associate\'s degree    260.966667    152.221344    144.846465
      bachelor\'s degree      94.333333     59.290909    158.426154
      high school            219.476190    231.485450    210.171264
      master\'s degree        112.500000    171.200000    157.571429
      some college            167.696429    236.352381    220.390592
      some high school        358.711111    284.874459    314.109788
male   associate\'s degree    244.125000    217.911765    211.751894
      bachelor\'s degree    161.527778    180.777778    223.170330
      high school           120.654545    234.892105    129.466132
      master\'s degree              NaN              NaN    126.060606
      some college           396.722222    233.922078    262.640000
      some high school       150.131868    186.495833     85.447619

race/ethnicity          group D      group E
gender parental level of education
female associate\'s degree    215.449275    192.970588
      bachelor\'s degree    214.858974    387.333333
      high school           181.441176    207.659091
      master\'s degree       228.780952    306.666667
      some college           110.063866    152.229167
      some high school       232.090000    387.766667
male   associate\'s degree    149.435385    226.147619
      bachelor\'s degree    203.095238    141.267857
      high school           158.396011    109.833333
      master\'s degree         47.071429              NaN
      some college           163.741935    144.368421
      some high school       173.156667    276.931818
```

1.0.34 以 `openml` 中的数据集 `1StudentPerformance` 为例，统计不同 **race** 和不同性别，不同的 **parental level of education** 下，样本的数量，并给出边际值（也是数量）。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

[3]: # 查看数据集的前五行

```
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

  test preparation course  math score  reading score  writing score
0              none        72           72           74
1      completed        69           90           88
2              none        90           95           93
3              none        47           57           44
4              none        76           78           75
```

[4]: # 数据透视表

```
pd.pivot_table(
    data,
    values="writing score",
    index=["gender", "parental level of education"],
    columns="race/ethnicity",
    aggfunc="count",
    margins=True,
    margins_name="样本总数"
)
```

```
[4]: race/ethnicity      group A  group B  group C  group D \
gender parental level of education
female associate\'s degree      6      23      45      24
      bachelor\'s degree      3      11      26      13
      high school      7      28      30      17
      master\'s degree      2       5       7      15
      some college      8      15      44      35
      some high school     10      22      28      25
male   associate\'s degree      8      18      33      26
      bachelor\'s degree      9       9      14      15
      high school     11      20      34      27
      master\'s degree      1       1      12       8
      some college     10      22      25      32
      some high school     14      16      21      25
样本总数      89      190      319      262

race/ethnicity      group E  样本总数
gender parental level of education
```

female	associate\'s degree	18	116
	bachelor\'s degree	10	63
	high school	12	94
	master\'s degree	7	36
	some college	16	118
	some high school	6	91
male	associate\'s degree	21	106
	bachelor\'s degree	8	55
	high school	10	102
	master\'s degree	1	23
	some college	19	108
	some high school	12	88
样本总数		140	1000

1.0.35 以 `openml` 中的数据集 `1StudentPerformance` 为例, 计算 `gender` 和 `race/ethnicity` 的二维列联表。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0              none          72          72          74
1        completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75
```

```
[4]: # 列联表
pd.crosstab(
    index=data["gender"],
    columns=data["race/ethnicity"]
)
```

```
[4]: race/ethnicity  group A  group B  group C  group D  group E
gender
female           36     104     180     129     69
male            53     86     139     133     71
```

1.0.36 以 openml 中的数据集中 **1StudentPerformance** 为例，计算 **gender** 和 **race/ethnicity** 和 **lunch** 的三维频率列联表，并给出边际值。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree  standard
1  female      group C      some college  standard
2  female      group B      master\'s degree  standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college  standard

   test preparation course  math score  reading score  writing score
0             none         72           72           74
1      completed         69           90           88
2             none         90           95           93
3             none         47           57           44
4             none         76           78           75
```

```
[4]: # 列联表
pd.crosstab(
    index=[data["gender"], data["lunch"]], # 必须是列表形式
    columns=data["race/ethnicity"],
```

```

        normalize=True,
        margins=True,
        margins_name="合计比例"
    )

```

```

[4]: race/ethnicity      group A  group B  group C  group D  group E  合计比例
gender lunch
female free/reduced    0.014    0.039    0.062    0.051    0.023  0.189
      standard         0.022    0.065    0.118    0.078    0.046  0.329
male   free/reduced    0.022    0.030    0.052    0.044    0.018  0.166
      standard         0.031    0.056    0.087    0.089    0.053  0.316
合计比例              0.089    0.190    0.319    0.262    0.140  1.000

```

1.0.37 以 `openml` 中的数据集中 `StudentPerformance` 为例，对数学成绩 `math score` 离散化分组。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1        completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75

```

```

[4]: # 离散变量分组
pd.cut(data["math score"], bins=10)

```



```
[4]: 0      (70.0, 80.0]
      1      (60.0, 70.0]
      2      (80.0, 90.0]
      3      (40.0, 50.0]
      4      (70.0, 80.0]
      ...
      995    (80.0, 90.0]
      996    (60.0, 70.0]
      997    (50.0, 60.0]
      998    (60.0, 70.0]
      999    (70.0, 80.0]
Name: math score, Length: 1000, dtype: category
Categories (10, interval[float64, right]): [(-0.1, 10.0] < (10.0, 20.0] < (20.0,
30.0] < (30.0, 40.0] ... (60.0, 70.0] < (70.0, 80.0] < (80.0, 90.0] < (90.0,
100.0]]
```

1.0.38 以 openml 中的数据集合 1StudentPerformance 为例，对 gender 进行虚拟变量化。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0              none          72          72          74
1        completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75
```

```
[4]: # 生成虚拟变量
dummy = pd.get_dummies(data["gender"], prefix="性别")
dummy
```

```
[4]:      性别 _female  性别 _male
0          1         0
1          1         0
2          1         0
3          0         1
4          0         1
..         ...         ...
995        1         0
996        0         1
997        1         0
998        1         0
999        1         0
```

[1000 rows x 2 columns]

```
[5]: # 从虚拟变量的 dataframe 转为正常的一列 dataframe
newdata = pd.from_dummies(dummy)
newdata
```

```
[5]:      性别 _female
0      性别 _female
1      性别 _female
2      性别 _female
3      性别 _male
4      性别 _male
..         ...
995  性别 _female
996  性别 _male
997  性别 _female
998  性别 _female
999  性别 _female
```

[1000 rows x 1 columns]

1.0.39 以 openml 中的数据集 1StudentPerformance 为例，对 gender 转为因子类型。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1        completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

```
[4]: # 变量因子化
newdata, index = pd.factorize(data["gender"])
newdata[:10]
```

```
[4]: array([0, 0, 0, 1, 1, 0, 0, 1, 1, 0], dtype=int64)
```

```
[5]: index
```

```
[5]: CategoricalIndex(['female', 'male'], categories=['female', 'male'],
    ordered=False, dtype='category')
```

1.0.40 以 openml 中的数据集 1StudentPerformance 为例，将宽数据转为长数据。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
```

```
)["frame"]
```

[3]: # 查看数据集的前五行

```
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

test preparation course  math score  reading score  writing score
0          none          72          72          74
1      completed          69          90          88
2          none          90          95          93
3          none          47          57          44
4          none          76          78          75
```

[4]: # 宽数据转为长数据

```
data.iloc[:5, [0,1,5,6]].melt(id_vars=["gender", "race/ethnicity"])
```

```
[4]:  gender race/ethnicity      variable  value
0  female      group B      math score      72
1  female      group C      math score      69
2  female      group B      math score      90
3   male      group A      math score      47
4   male      group C      math score      76
5  female      group B      reading score     72
6  female      group C      reading score     90
7  female      group B      reading score     95
8   male      group A      reading score     57
9   male      group C      reading score     78
```

1.0.41 以 openml 中的数据集 1StudentPerformance 为例, 将 gender 的字符串全部大写, 将 race/ethnicity 的字符串全部小写。

[1]: # 导入数据集获取工具

```
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

[2]: # 获取数据

```
data = fetch_openml(
    data_id=43255,
    as_frame=True,
```

```
parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

  test preparation course  math score  reading score  writing score
0                none         72           72           74
1             completed         69           90           88
2                none         90           95           93
3                none         47           57           44
4                none         76           78           75
```

```
[4]: # 字符串大小写
data["gender"].str.upper()
```

```
[4]: 0      FEMALE
1      FEMALE
2      FEMALE
3      MALE
4      MALE
...
995    FEMALE
996     MALE
997    FEMALE
998    FEMALE
999    FEMALE
Name: gender, Length: 1000, dtype: object
```

```
[5]: data["race/ethnicity"].str.lower()
```

```
[5]: 0      group b
1      group c
2      group b
3      group a
4      group c
...
995    group e
996    group c
997    group c
```

```

998     group d
999     group d
Name: race/ethnicity, Length: 1000, dtype: object

```

1.0.42 以 `openml` 中的数据集 `1StudentPerformance` 为例, 计算 `lunch` 列字符串中字符的个数。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

```

```

      test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75

```

```

[4]: # 字符个数
data["lunch"].str.len()

```

```

[4]: 0      8
     1      8
     2      8
     3     12
     4      8
     ..
    995      8
    996     12

```

```

997    12
998     8
999    12
Name: lunch, Length: 1000, dtype: int64

```

1.0.43 以 openml 中的数据集合 1StudentPerformance 为例，将 race/ethnicity 中的字符串空格去除。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor's degree      standard
1  female      group C              some college      standard
2  female      group B      master's degree      standard
3   male      group A      associate's degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75

```

```

[4]: # 去除空格
data["race/ethnicity"].str.replace(" ", "")

```

```

[4]: 0      groupB
1      groupC
2      groupB
3      groupA
4      groupC
...
995    groupE

```

```

996    groupC
997    groupC
998    groupD
999    groupD
Name: race/ethnicity, Length: 1000, dtype: object

```

1.0.44 以 `openml` 中的数据集 `1StudentPerformance` 为例，将 `race/ethnicity` 的字符串按照空格分割成列表。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1        completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75

```

```

[4]: # 分割为列表
strlist = data["race/ethnicity"].str.split(" ")
strlist

```

```

[4]: 0    [group, B]
     1    [group, C]
     2    [group, B]
     3    [group, A]

```



```

4      [group, C]
      ...
995    [group, E]
996    [group, C]
997    [group, C]
998    [group, D]
999    [group, D]
Name: race/ethnicity, Length: 1000, dtype: object

```

```
[5]: # 获取分割的列表的第二个元素
      strlist.str[1]
```

```

[5]: 0      B
      1      C
      2      B
      3      A
      4      C
      ..
995    E
996    C
997    C
998    D
999    D
Name: race/ethnicity, Length: 1000, dtype: object

```

1.0.45 以 `openml` 中的数据集合 `1StudentPerformance` 为例，将 `race/ethnicity` 的字符串按照空格分割成列表，并将列表中的各个元素作为列加入到 `dataframe` 中。

```
[1]: # 导入数据集获取工具
      from sklearn.datasets import fetch_openml
      # 导入数据分析库
      import pandas as pd
```

```
[2]: # 获取数据
      data = fetch_openml(
          data_id=43255,
          as_frame=True,
          parser="pandas"
      )["frame"]
```

```
[3]: # 查看数据集的前五行
      data.head()
```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard

```

2	female	group B	master\'s degree	standard
3	male	group A	associate\'s degree	free/reduced
4	male	group C	some college	standard

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 分割为列表
strlist = data["race/ethnicity"].str.split(" ", expand=True)
strlist
```

```
[4]:      0  1
0   group B
1   group C
2   group B
3   group A
4   group C
..    .. ..
995 group E
996 group C
997 group C
998 group D
999 group D
```

[1000 rows x 2 columns]

1.0.46 以 `openml` 中的数据集 `1StudentPerformance` 为例，将 `parental level of education` 的字符串中的反斜杠去掉。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor's degree      standard
1  female      group C              some college      standard
2  female      group B      master's degree      standard
3   male      group A      associate's degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

```
[4]: # 去掉反斜杠, 使用替换
data["parental level of education"].iloc[0]
```

```
[4]: "bachelor\\'s degree"
```

```
[5]: # 使用正则表达式替换
data["parental level of education"].str.replace("\\", "", regex=True)
```

```
[5]: 0      bachelor's degree
1      some college
2      master's degree
3      associate's degree
4      some college
...
995     master's degree
996      high school
997      high school
998      some college
999      some college
Name: parental level of education, Length: 1000, dtype: object
```

1.0.47 以 `openml` 中的数据集合 `1StudentPerformance` 为例, 将 `math score`, `reading score` 和 `writing score` 三列合并为一列, 以逗号分隔。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

```
[4]: # 列之间的合并
newS = data["math score"].astype("string").str.cat(data["reading score"].
    ↳ astype("string"), sep=",").str.cat(data["writing score"].astype("string"),
    ↳ sep=",")
newS
```

```
[4]: 0      72,72,74
1      69,90,88
2      90,95,93
3      47,57,44
4      76,78,75
...
995    88,99,95
996    62,55,55
997    59,71,65
998    68,78,77
999    77,86,86
Name: math score, Length: 1000, dtype: string
```

1.0.48 对含有缺失值的列进行求和和求乘积，看看结果如何

```
[1]: import pandas as pd
import numpy as np
```

```
[2]: # 生成 dataframe
df = pd.DataFrame(
    np.random.randn(5, 3),
    index=["a", "c", "e", "f", "h"],
    columns=["one", "two", "three"],
)
df["four"] = "bar"
df["five"] = df["one"] > 0
df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])
df
```

```
[2]:
```

	one	two	three	four	five
a	0.271842	-0.350975	0.194960	bar	True
b	NaN	NaN	NaN	NaN	NaN
c	-0.104840	-0.469391	-0.410844	bar	False
d	NaN	NaN	NaN	NaN	NaN
e	-0.059557	0.436472	-0.792940	bar	False
f	0.575772	-0.903377	0.192159	bar	True
g	NaN	NaN	NaN	NaN	NaN
h	-1.555696	0.607744	-0.759766	bar	False

```
[3]: # 计算含有缺失值的列的和，默认是跳过缺失值
df["one"].sum()
```

```
[3]: -0.8724777568039623
```

```
[4]: # 不跳过缺失值，那么得到的和就是 NaN
df["two"].sum(skipna=False)
```

```
[4]: nan
```

```
[5]: # 计算含有缺失值的列的积，默认是跳过缺失值
df["one"].prod()
```

```
[5]: -0.0015203634461877212
```

```
[6]: # 不跳过缺失值，那么得到的积就是 NaN
df["two"].prod(skipna=False)
```

```
[6]: nan
```

1.0.49 对含有缺失值的列进行填充，将缺失值以其他的值替换

```
[1]: import pandas as pd
import numpy as np
```

```
[2]: # 生成 dataframe
df = pd.DataFrame(
    np.random.randn(5, 3),
    index=["a", "c", "e", "f", "h"],
    columns=["one", "two", "three"],
)
df["four"] = "bar"
df["five"] = df["one"] > 0
df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])
df
```

```
[2]:
```

	one	two	three	four	five
a	1.010471	1.775946	0.116074	bar	True
b	NaN	NaN	NaN	NaN	NaN
c	-0.709978	0.945866	1.369733	bar	False
d	NaN	NaN	NaN	NaN	NaN
e	0.010790	0.209315	1.177950	bar	True
f	0.028884	1.194446	-0.041722	bar	True
g	NaN	NaN	NaN	NaN	NaN
h	1.731725	0.499953	-0.015604	bar	True

```
[3]: # 以均值替换
df["one"].fillna(df["one"].mean())
```

```
[3]: a    1.010471
b    0.414378
c   -0.709978
d    0.414378
e    0.010790
f    0.028884
g    0.414378
h    1.731725
Name: one, dtype: float64
```

1.0.50 对含有缺失值的列或者行删除。

```
[1]: import pandas as pd
import numpy as np
```

```
[2]: # 生成 dataframe
df = pd.DataFrame(
    np.random.randn(5, 3),
```

```

index=["a", "c", "e", "f", "h"],
columns=["one", "two", "three"],
)
df["four"] = "bar"
df["five"] = df["one"] > 0
df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])
df

```

```

[2]:
      one      two      three four  five
a  1.105184  0.696500  1.024294  bar   True
b         NaN         NaN         NaN  NaN   NaN
c  0.286259 -1.286451  1.746801  bar   True
d         NaN         NaN         NaN  NaN   NaN
e  0.283374 -0.699082  1.299307  bar   True
f -0.698294  0.484663  1.120755  bar  False
g         NaN         NaN         NaN  NaN   NaN
h -2.119560  0.173416  0.700130  bar  False

```

```

[3]: # 删除含有缺失值的行，存在缺失值就删除
df.dropna(axis=0, how="any")

```

```

[3]:
      one      two      three four  five
a  1.105184  0.696500  1.024294  bar   True
c  0.286259 -1.286451  1.746801  bar   True
e  0.283374 -0.699082  1.299307  bar   True
f -0.698294  0.484663  1.120755  bar  False
h -2.119560  0.173416  0.700130  bar  False

```

```

[4]: # 删除含有缺失值的列，全部都是缺失值才删除
df.dropna(axis=1, how="all")

```

```

[4]:
      one      two      three four  five
a  1.105184  0.696500  1.024294  bar   True
b         NaN         NaN         NaN  NaN   NaN
c  0.286259 -1.286451  1.746801  bar   True
d         NaN         NaN         NaN  NaN   NaN
e  0.283374 -0.699082  1.299307  bar   True
f -0.698294  0.484663  1.120755  bar  False
g         NaN         NaN         NaN  NaN   NaN
h -2.119560  0.173416  0.700130  bar  False

```

1.0.51 对含有缺失值的列进行计数，求非缺失值的个数。

```

[1]: import pandas as pd
import numpy as np

```

```
[2]: # 生成 dataframe
df = pd.DataFrame(
    np.random.randn(5, 3),
    index=["a", "c", "e", "f", "h"],
    columns=["one", "two", "three"],
)
df["four"] = "bar"
df["five"] = df["one"] > 0
df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])
df
```

```
[2]:
```

	one	two	three	four	five
a	-0.126636	-1.491391	0.700825	bar	False
b	NaN	NaN	NaN	NaN	NaN
c	0.981905	-1.506363	0.082841	bar	True
d	NaN	NaN	NaN	NaN	NaN
e	-0.383442	1.491860	-0.551062	bar	False
f	-0.182666	0.236521	0.678531	bar	False
g	NaN	NaN	NaN	NaN	NaN
h	0.984269	-0.652942	-1.957255	bar	True

```
[3]: # 求非缺失值的个数
df["one"].count()
```

```
[3]: 5
```

1.0.52 对含有缺失值的列进行插值。

```
[1]: import pandas as pd
import numpy as np
```

```
[2]: # 生成 dataframe
df = pd.DataFrame(
    np.random.randn(5, 3),
    index=["a", "c", "e", "f", "h"],
    columns=["one", "two", "three"],
)
df["four"] = "bar"
df["five"] = df["one"] > 0
df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])
df
```

```
[2]:
```

	one	two	three	four	five
a	0.957477	0.356867	1.238439	bar	True
b	NaN	NaN	NaN	NaN	NaN
c	-1.194969	-0.859723	-2.106189	bar	False
d	NaN	NaN	NaN	NaN	NaN


```
e 0.327321 0.633526 -0.024933 bar True
f 1.027312 0.586582 0.242355 bar True
g      NaN      NaN      NaN NaN NaN
h 0.095649 0.213300 0.137504 bar True
```

```
[3]: # 线性插值
df["one"].interpolate()
```

```
[3]: a    0.957477
     b   -0.118746
     c   -1.194969
     d   -0.433824
     e    0.327321
     f    1.027312
     g    0.561480
     h    0.095649
     Name: one, dtype: float64
```

1.0.53 以 `openml` 中的数据集 `1StudentPerformance` 为例，将 `gender` 变量变为 `category` 类型。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
```

3	none	47	57	44
4	none	76	78	75

```
[4]: # 将字符串变量 category 化
data["gender"].astype("category")
```

```
[4]: 0    female
      1    female
      2    female
      3     male
      4     male
      ...
     995   female
     996     male
     997   female
     998   female
     999   female
Name: gender, Length: 1000, dtype: category
Categories (2, object): ['female', 'male']
```

1.0.54 以 openml 中的数据集 1StudentPerformance 为例，将 gender 变量变为 category 类型并修改类别名称为”男性”和”女性”。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

test preparation course  math score  reading score  writing score
```

0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 将 gender 修改为分类变量
newvar = data["gender"].astype("category")
newvar.head()
```

```
[4]: 0    female
1    female
2    female
3     male
4     male
Name: gender, dtype: category
Categories (2, object): ['female', 'male']
```

```
[5]: # 查看分类变量的类别名称
print(newvar.cat.categories)
```

```
Index(['female', 'male'], dtype='object')
```

```
[6]: # 修改分类变量的类别名称
new_categories = ["女性", "男性"]
newvar = newvar.cat.rename_categories(new_categories)
newvar.head()
```

```
[6]: 0    女性
1    女性
2    女性
3    男性
4    男性
Name: gender, dtype: category
Categories (2, object): ['女性', '男性']
```

1.055 以 openml 中的数据集合 1StudentPerformance 为例，将 gender 变量变为 category 类型并添加一个类别“Unknown”。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
```

```
data_id=43255,
as_frame=True,
parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C      some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C      some college      standard

  test preparation course  math score  reading score  writing score
0              none        72           72           74
1      completed        69           90           88
2              none        90           95           93
3              none        47           57           44
4              none        76           78           75
```

```
[4]: # 将 gender 修改为分类变量
newvar = data["gender"].astype("category")
newvar.head()
```

```
[4]: 0  female
1  female
2  female
3   male
4   male
Name: gender, dtype: category
Categories (2, object): ['female', 'male']
```

```
[5]: # 添加一个类别
newvar = newvar.cat.add_categories(["Unknown"])
newvar.head()
```

```
[5]: 0  female
1  female
2  female
3   male
4   male
Name: gender, dtype: category
Categories (3, object): ['female', 'male', 'Unknown']
```

1.0.56 以 `openml` 中的数据集 `1StudentPerformance` 为例，将 `gender` 变量变为 `category` 类型并添加一个类别“Unknown”，然后删除该类别。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75
```

```
[4]: # 将 gender 修改为分类变量
newvar = data["gender"].astype("category")
newvar.head()
```

```
[4]: 0    female
1    female
2    female
3     male
4     male
Name: gender, dtype: category
Categories (2, object): ['female', 'male']
```

```
[5]: # 添加一个类别
newvar = newvar.cat.add_categories(["Unknown"])
newvar.head()
```

```
[5]: 0    female
      1    female
      2    female
      3     male
      4     male
      Name: gender, dtype: category
      Categories (3, object): ['female', 'male', 'Unknown']
```

```
[6]: newvar.cat.remove_categories(["Unknown"])
```

```
[6]: 0    female
      1    female
      2    female
      3     male
      4     male
      ...
     995    female
     996     male
     997    female
     998    female
     999    female
      Name: gender, Length: 1000, dtype: category
      Categories (2, object): ['female', 'male']
```

```
[7]: newvar.cat.remove_categories(["Unknown", "female"])
```

```
[7]: 0     NaN
      1     NaN
      2     NaN
      3    male
      4    male
      ...
     995     NaN
     996    male
     997     NaN
     998     NaN
     999     NaN
      Name: gender, Length: 1000, dtype: category
      Categories (1, object): ['male']
```

1.0.57 以 `openml` 中的数据集 `1StudentPerformance` 为例，将 `gender` 变量变为 `category` 类型并添加一个类别“Unknown”，然后删除没用的类别（类别无变量值对应）。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
[3]: # 查看数据集的前五行
data.head()
```

```
[3]:  gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

test preparation course  math score  reading score  writing score
0              none          72          72          74
1      completed          69          90          88
2              none          90          95          93
3              none          47          57          44
4              none          76          78          75
```

```
[4]: # 将 gender 修改为分类变量
newvar = data["gender"].astype("category")
newvar.head()
```

```
[4]: 0    female
1    female
2    female
3     male
4     male
Name: gender, dtype: category
Categories (2, object): ['female', 'male']
```

```
[5]: # 添加一个类别
newvar = newvar.cat.add_categories(["Unknown"])
newvar.head()
```

```
[5]: 0    female
      1    female
      2    female
      3     male
      4     male
      Name: gender, dtype: category
      Categories (3, object): ['female', 'male', 'Unknown']
```

```
[6]: # 删除无用类别
      newvar = newvar.cat.remove_unused_categories()
      newvar
```

```
[6]: 0     female
      1     female
      2     female
      3      male
      4      male
      ...
     995    female
     996      male
     997    female
     998    female
     999    female
      Name: gender, Length: 1000, dtype: category
      Categories (2, object): ['female', 'male']
```

1.0.58 以 openml 中的数据集 1StudentPerformance 为例，将变量 parental level of education 变为有序的 category 类型。

```
[1]: # 导入数据集获取工具
      from sklearn.datasets import fetch_openml
      # 导入数据分析库
      import pandas as pd
```

```
[2]: # 获取数据
      data = fetch_openml(
          data_id=43255,
          as_frame=True,
          parser="pandas"
      )["frame"]
```

```
[3]: # 查看数据集的前五行
      data.head()
```

```
[3]:   gender race/ethnicity parental level of education    lunch \
0  female      group B      bachelor\'s degree    standard
1  female      group C          some college    standard
```


2	female	group B	master\'s degree	standard
3	male	group A	associate\'s degree	free/reduced
4	male	group C	some college	standard

	test preparation course	math score	reading score	writing score
0	none	72	72	74
1	completed	69	90	88
2	none	90	95	93
3	none	47	57	44
4	none	76	78	75

```
[4]: # 将 gender 修改为分类变量
t = pd.CategoricalDtype([
    "some high school", "high school", "some college",
    "associate\\'s degree", "bachelor\\'s degree", "master\\'s degree"
],
    ordered=True # 从小到大排序
)
newvar = data["parental level of education"].astype(t)
newvar.head()
```

```
[4]: 0    bachelor\'s degree
1      some college
2    master\'s degree
3    associate\'s degree
4      some college
Name: parental level of education, dtype: category
Categories (6, object): ['some high school' < 'high school' < 'some college' <
'associate\'s degree' < 'bachelor\'s degree' < 'master\'s degree']
```

1.0.59 以 openml 中的数据集中 StudentPerformance 为例，将变量 parental level of education 变为有序的 category 类型并且修改类别之间的顺序。

```
[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd
```

```
[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]
```

[3]: # 查看数据集的前五行

```
data.head()
```

```
[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C              some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C              some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1      completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75
```

[4]: # 将 *gender* 修改为分类变量

```
t = pd.CategoricalDtype([
    "some high school", "high school", "some college",
    "associate\\'s degree", "bachelor\\'s degree", "master\\'s degree"
],
    ordered=True # 从小到大排序
)
newvar = data["parental level of education"].astype(t)
newvar.head()
```

```
[4]: 0      bachelor\'s degree
1      some college
2      master\'s degree
3      associate\'s degree
4      some college
Name: parental level of education, dtype: category
Categories (6, object): ['some high school' < 'high school' < 'some college' <
'some college' < 'associate\'s degree' < 'bachelor\'s degree' < 'master\'s degree']
```

```
[6]: newvar = data["parental level of education"].cat.reorder_categories([
    "high school", "some high school", "associate\\'s degree",
    "some college", "bachelor\\'s degree", "master\\'s degree"
],
    ordered=True # 从小到大排序
)
newvar.head()
```

```
[6]: 0      bachelor\'s degree
1      some college
```

```

2      master\'s degree
3      associate\'s degree
4      some college
Name: parental level of education, dtype: category
Categories (6, object): ['high school' < 'some high school' < 'associate\'s
degree' < 'some college' < 'bachelor\'s degree' < 'master\'s degree']

```

1.0.60 以 `openml` 中的数据集 `1StudentPerformance` 为例，将变量 `parental level of education` 变为有序的 `category` 类型并且对该列和 `math score` 作为因子排序。

```

[1]: # 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入数据分析库
import pandas as pd

```

```

[2]: # 获取数据
data = fetch_openml(
    data_id=43255,
    as_frame=True,
    parser="pandas"
)["frame"]

```

```

[3]: # 查看数据集的前五行
data.head()

```

```

[3]:   gender race/ethnicity parental level of education      lunch \
0  female      group B      bachelor\'s degree      standard
1  female      group C          some college      standard
2  female      group B      master\'s degree      standard
3   male      group A      associate\'s degree  free/reduced
4   male      group C          some college      standard

   test preparation course  math score  reading score  writing score
0              none         72           72           74
1        completed         69           90           88
2              none         90           95           93
3              none         47           57           44
4              none         76           78           75

```

```

[4]: # 将 gender 修改为分类变量
t = pd.CategoricalDtype([
    "some high school", "high school", "some college",
    "associate\'s degree", "bachelor\'s degree", "master\'s degree"
],
    ordered=True # 从小到大排序
)

```

```
data["parental level of education"] = data["parental level of education"].astype(t)
data["parental level of education"].head()
```

```
[4]: 0    bachelor\'s degree
      1         some college
      2    master\'s degree
      3  associate\'s degree
      4         some college
      Name: parental level of education, dtype: category
      Categories (6, object): ['some high school' < 'high school' < 'some college' <
      'associate\'s degree' < 'bachelor\'s degree' < 'master\'s degree']
```

```
[5]: # 排序
      data.sort_values(by=["parental level of education", "math score"], ascending=False)
```

```
[5]:      gender race/ethnicity parental level of education      lunch \
618    male      group D      master\'s degree      standard
685  female      group E      master\'s degree      standard
957  female      group D      master\'s degree      standard
846    male      group C      master\'s degree      standard
2    female      group B      master\'s degree      standard
..      ...      ...      ...      ...
683  female      group C      some high school  free/reduced
363  female      group D      some high school  free/reduced
338  female      group B      some high school  free/reduced
17   female      group B      some high school  free/reduced
59   female      group C      some high school  free/reduced

      test preparation course  math score  reading score  writing score
618                none        95            81            84
685            completed        94            99           100
957                none        92           100           100
846            completed        91            85            85
2                none        90            95            93
..      ...      ...      ...      ...
683            completed        29            40            44
363                none        27            34            32
338                none        24            38            27
17                none        18            32            28
59                none         0             17            10
```

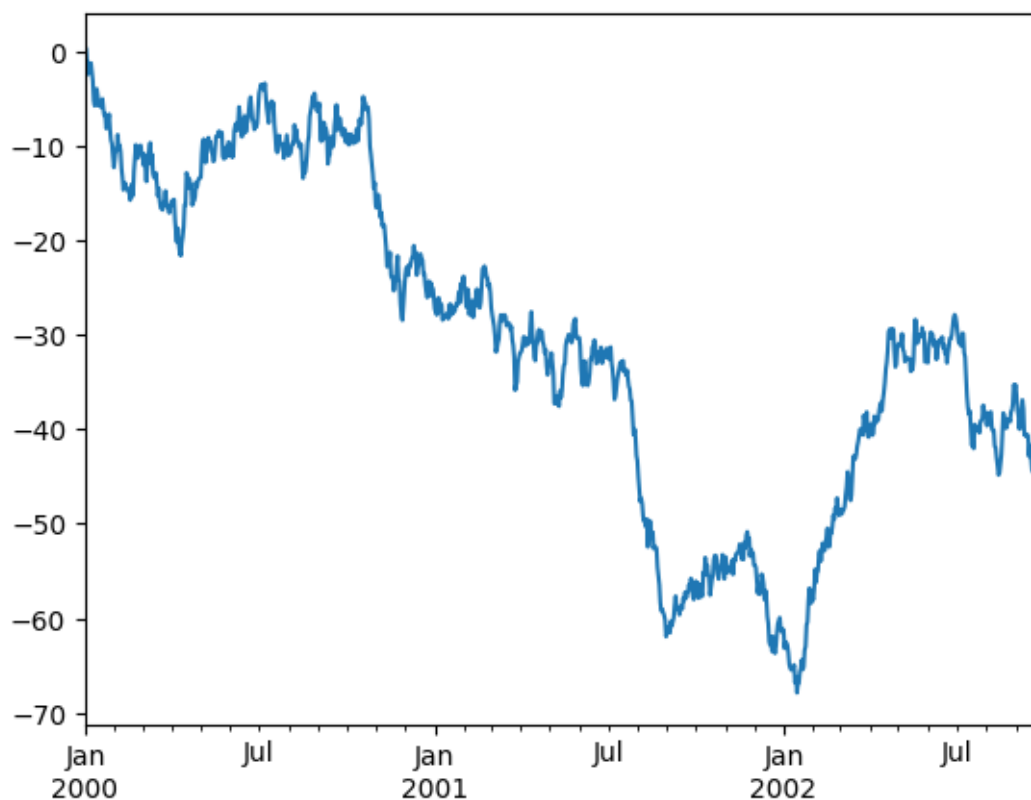
```
[1000 rows x 8 columns]
```

1.0.61 绘制线图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
ts = pd.Series(
    np.random.randn(1000), # 数字
    index=pd.date_range("1/1/2000", periods=1000) # 时间
)
# 累加
ts = ts.cumsum()
ts.plot()
```

[2]: <Axes: >

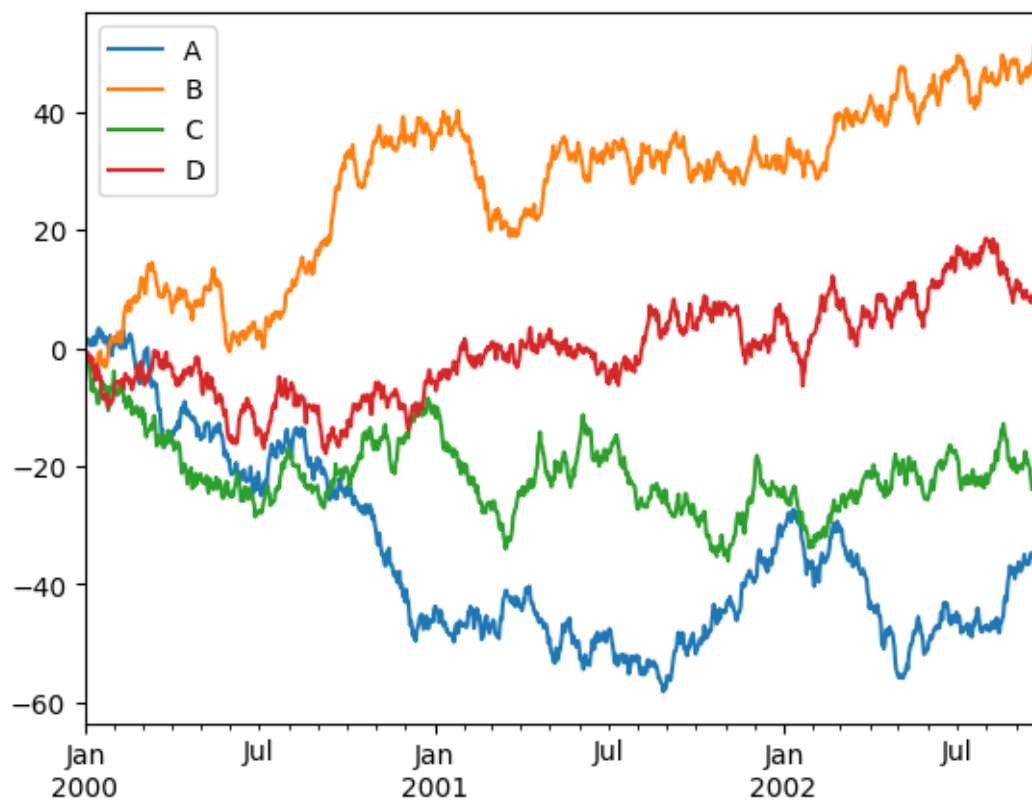


1.0.62 绘制多列数据的线图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[3]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    np.random.randn(1000, 4),
    index=pd.date_range("1/1/2000", periods=1000),
    columns=list("ABCD")
)
# 累加
df = df.cumsum()
df.plot()
```

[3]: <Axes: >

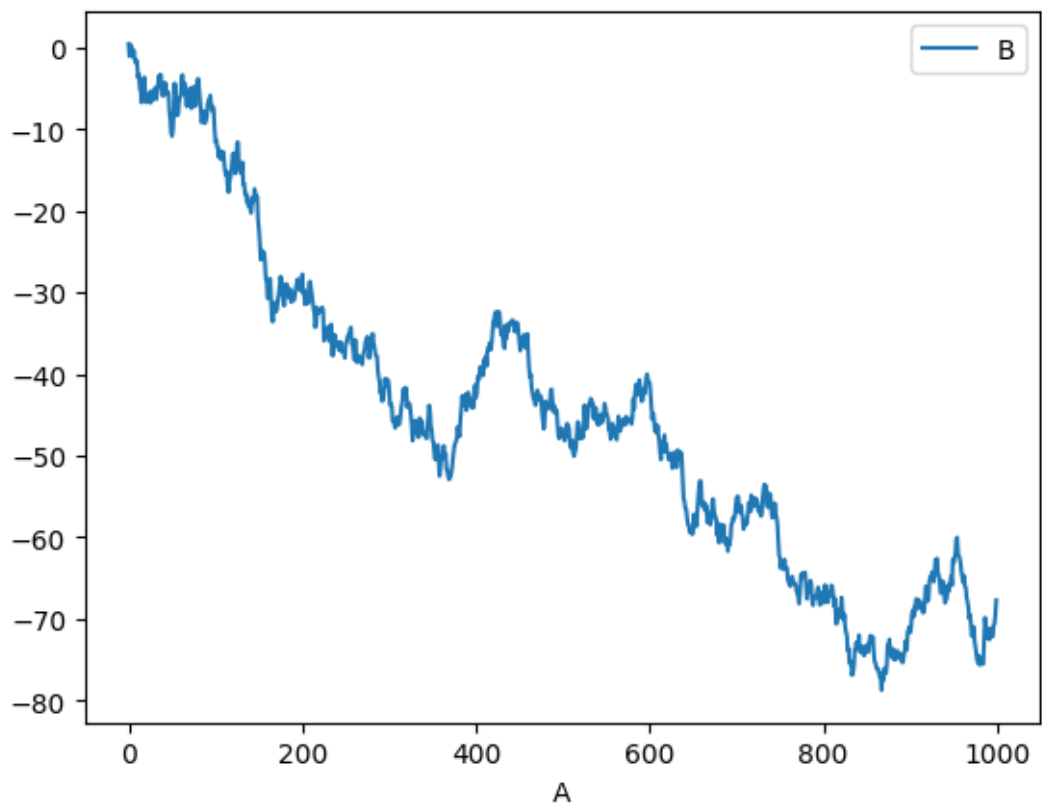


1.0.63 绘制多列数据中某两列的线图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.randn(1000, 2), columns=["B", "C"]).cumsum()
# 修改 A 列的值
df["A"] = pd.Series(list(range(len(df))))
df.plot(x="A", y="B")
```

[2]: <Axes: xlabel='A'>

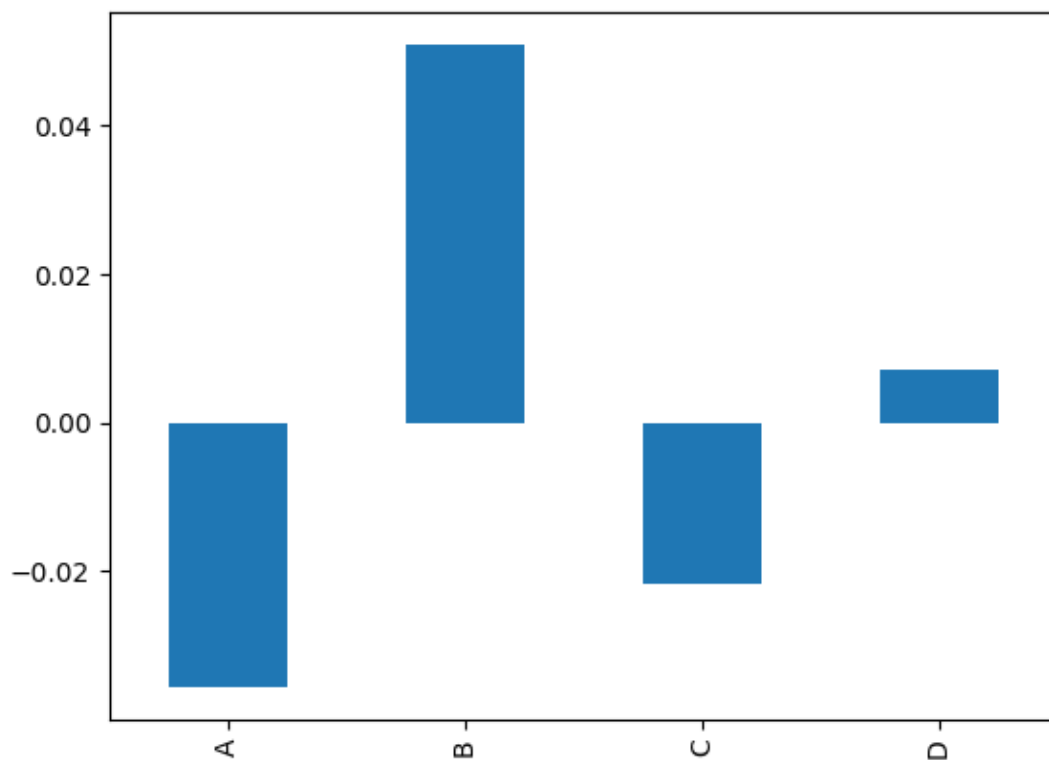


1.0.64 绘制多列数据的均值柱状图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    np.random.randn(1000, 4),
    index=pd.date_range("1/1/2000", periods=1000),
    columns=list("ABCD")
)
# 累加
df = df.mean()
df.plot(kind="bar")
```

[2]: <Axes: >



1.0.65 绘制多列数据的柱状图，分组柱状图

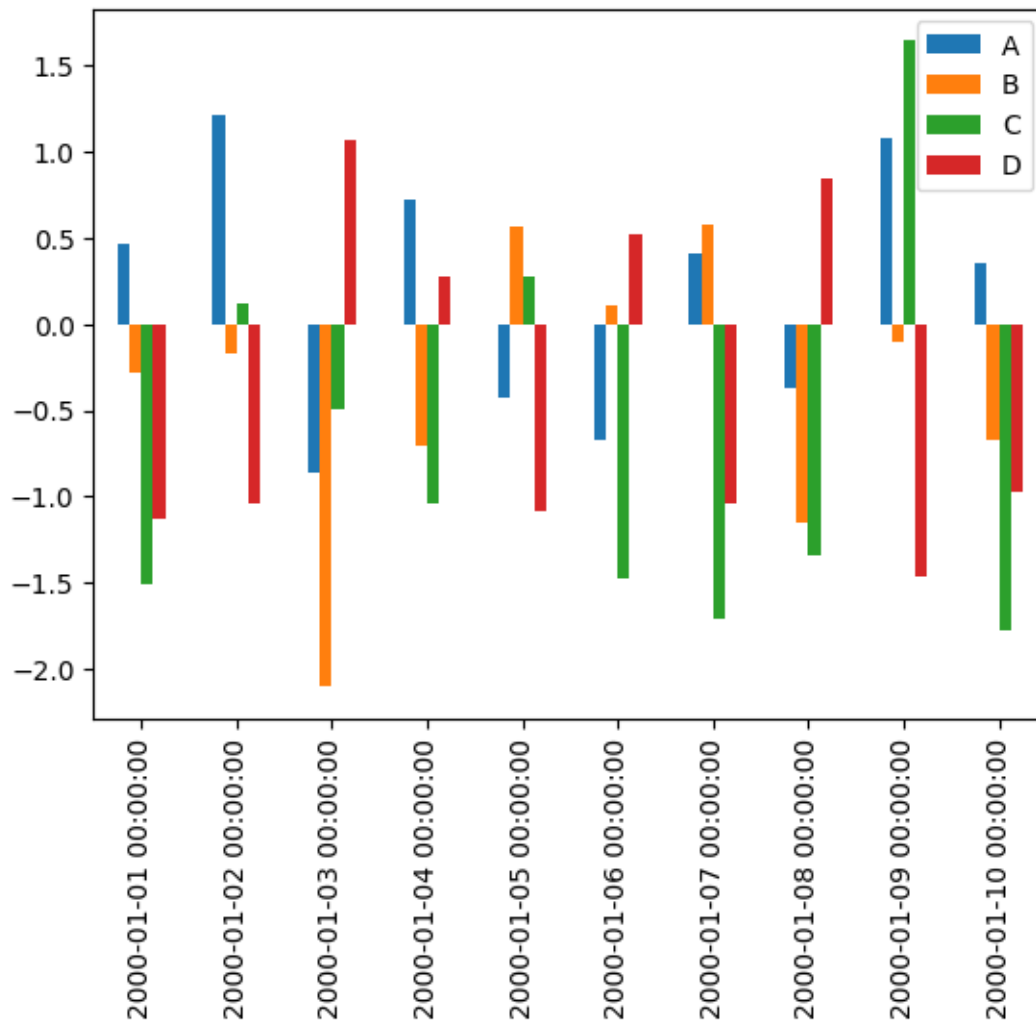
```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[3]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
```



```
df = pd.DataFrame(  
    np.random.randn(10, 4),  
    index=pd.date_range("1/1/2000", periods=10),  
    columns=list("ABCD")  
)  
df.plot(kind="bar")
```

[3]: <Axes: >

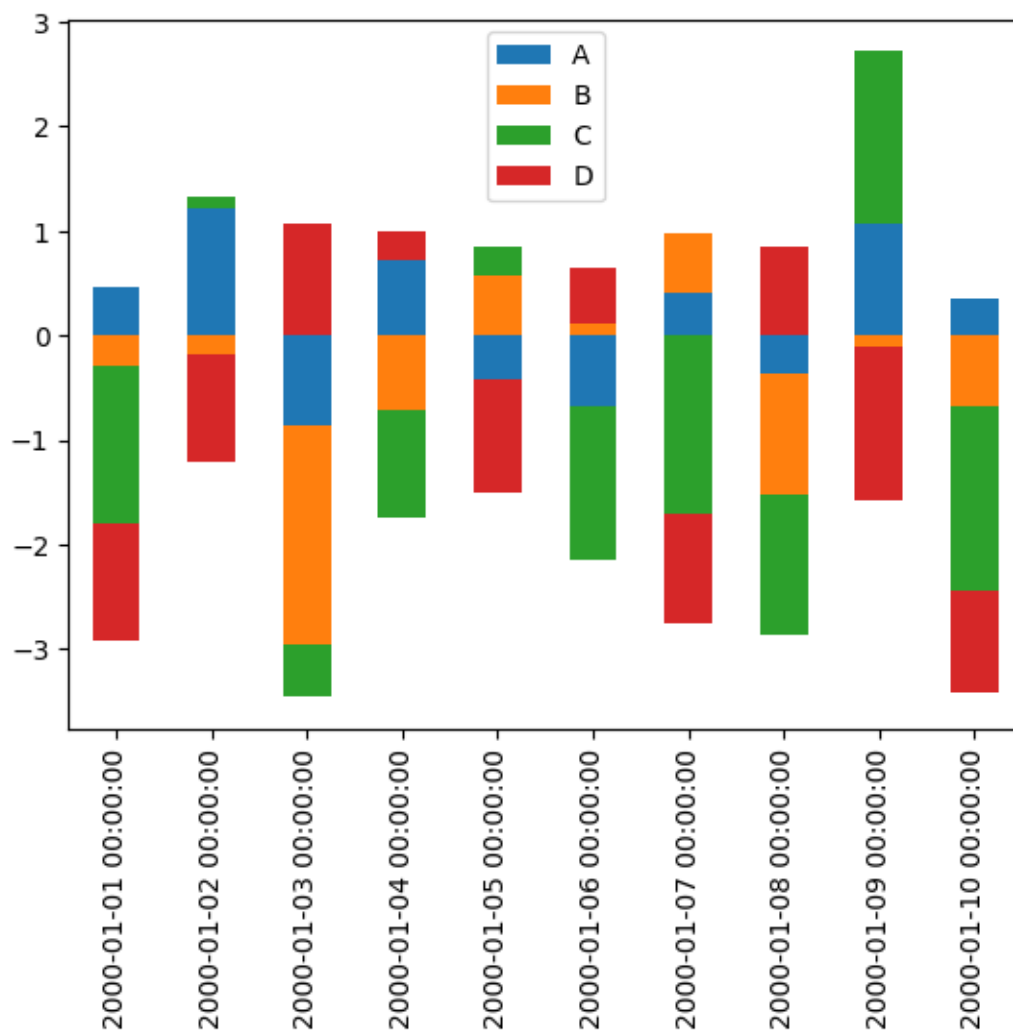


1.0.66 绘制多列数据的柱状图，堆叠柱状图

```
[1]: # 导入基础计算库  
import numpy as np  
# 导入数据分析库  
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    np.random.randn(10, 4),
    index=pd.date_range("1/1/2000", periods=10),
    columns=list("ABCD")
)
df.plot(kind="bar", stacked=True)
```

[2]: <Axes: >

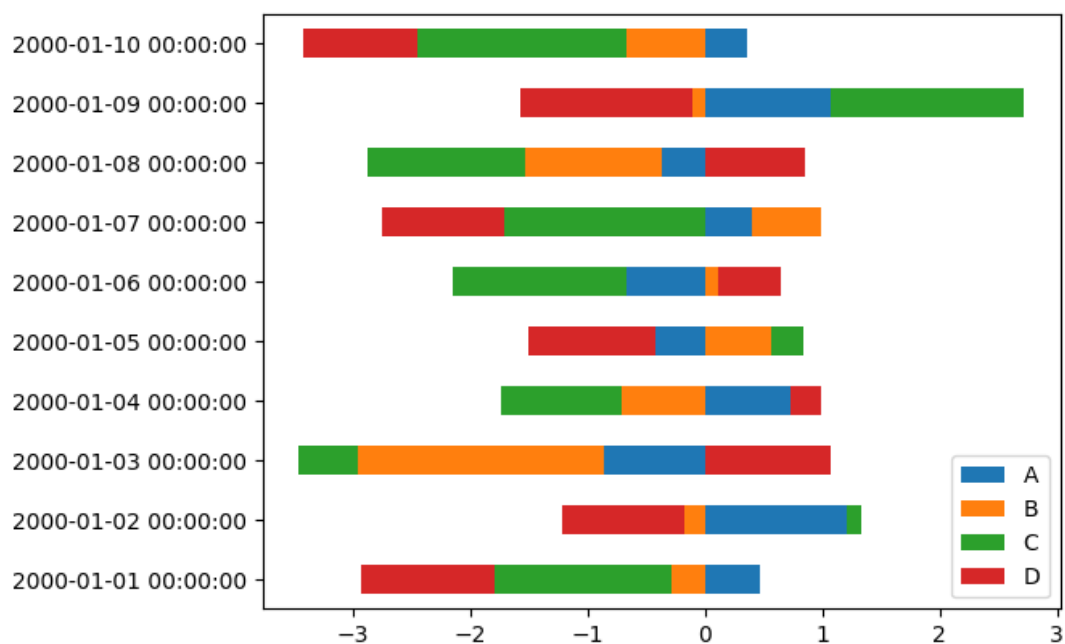


1.0.67 绘制多列数据的水平柱状图，堆叠柱状图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    np.random.randn(10, 4),
    index=pd.date_range("1/1/2000", periods=10),
    columns=list("ABCD")
)
df.plot(kind="barh", stacked=True)
```

[2]: <Axes: >

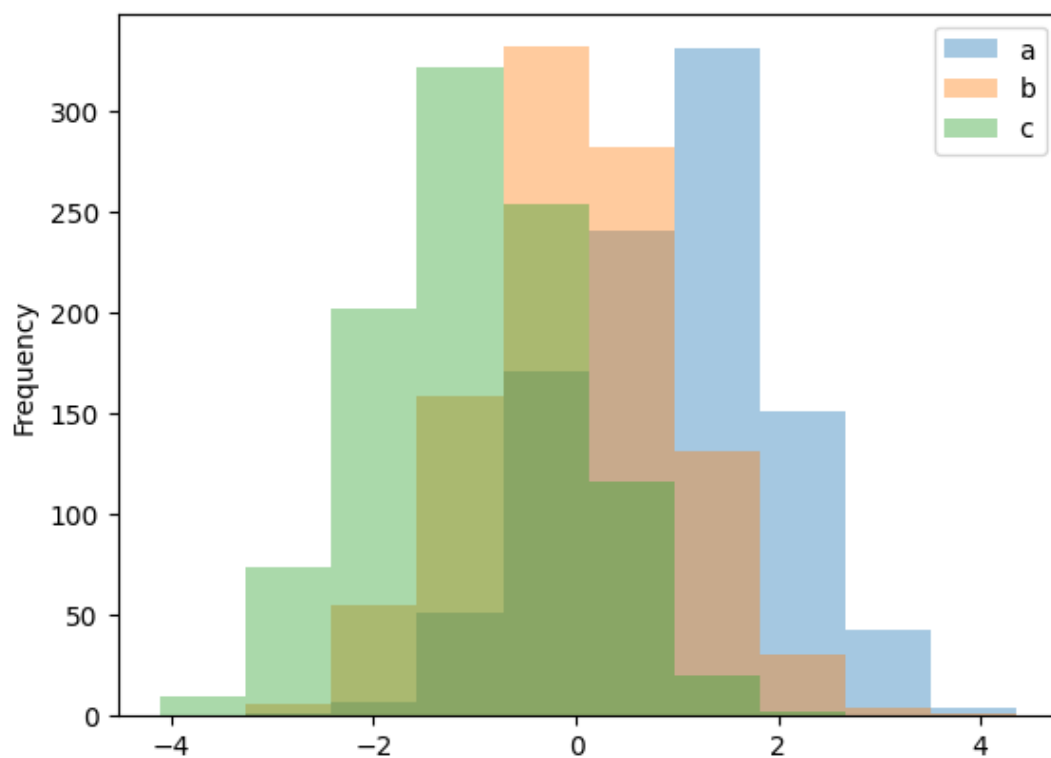


1.0.68 绘制多列数据的直方图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    {
        "a": np.random.randn(1000) + 1,
        "b": np.random.randn(1000),
        "c": np.random.randn(1000) - 1,
    },
    columns=["a", "b", "c"]
)
df.plot(kind="hist", alpha=0.4)
```

[2]: <Axes: ylabel='Frequency'>

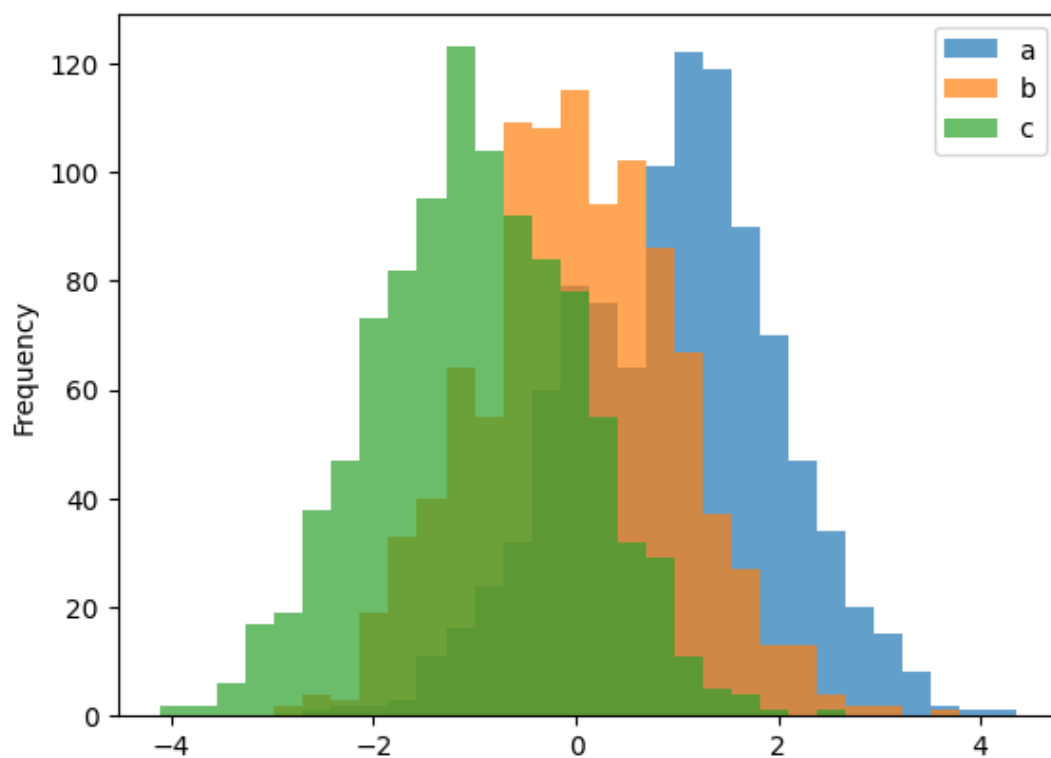


1.0.69 绘制多列数据的直方图，指定组数为 30

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    {
        "a": np.random.randn(1000) + 1,
        "b": np.random.randn(1000),
        "c": np.random.randn(1000) - 1,
    },
    columns=["a", "b", "c"]
)
df.plot(kind="hist", alpha=0.7, bins=30)
```

[2]: <Axes: ylabel='Frequency'>

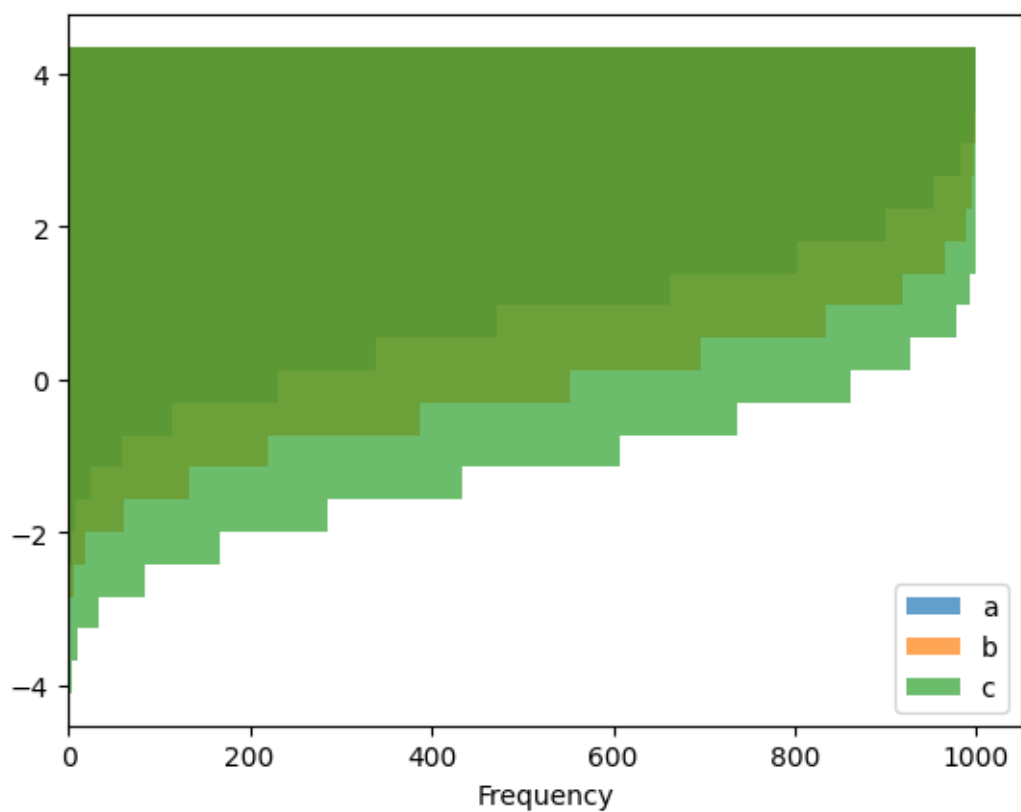


1.0.70 绘制多列数据的水平累积直方图，组数为 20

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    {
        "a": np.random.randn(1000) + 1,
        "b": np.random.randn(1000),
        "c": np.random.randn(1000) - 1,
    },
    columns=["a", "b", "c"]
)
df.plot(kind="hist", alpha=0.7, bins=20, orientation="horizontal", cumulative=True)
```

[2]: <Axes: xlabel='Frequency'>

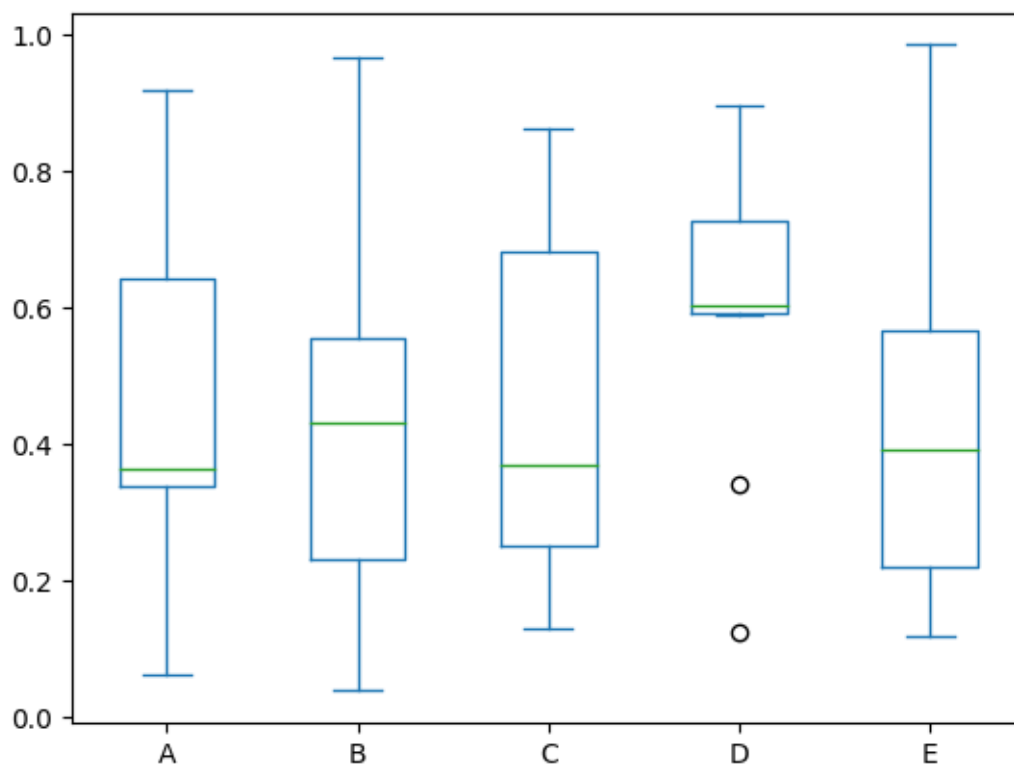


1.0.71 绘制多列数据的箱线图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[3]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D", "E"])
df.plot(kind="box")
```

[3]: <Axes: >



1.0.72 绘制多列数据的箱线图，给点外观颜色

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

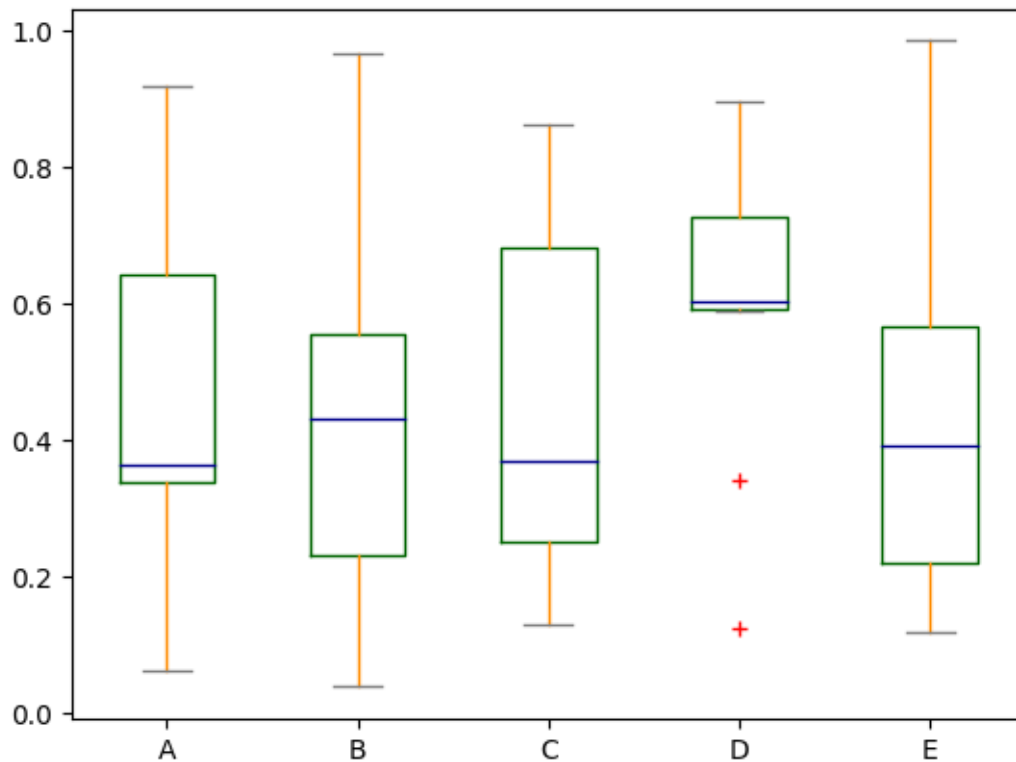
```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D", "E"])
# 设置相关颜色
color = {
    "boxes": "DarkGreen",
    "whiskers": "DarkOrange",
```

```

    "medians": "DarkBlue",
    "caps": "Gray",
}
df.plot(kind="box",color=color, sym="r+")

```

[2]: <Axes: >



1.0.73 绘制多列数据的水平箱线图

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

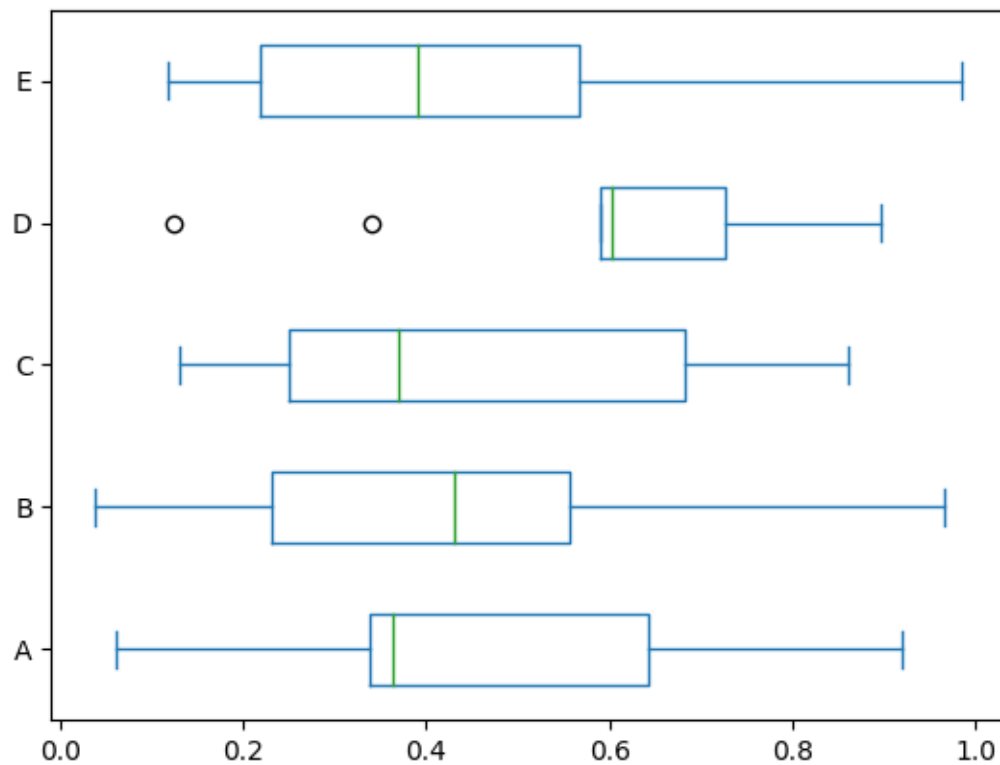
```

```

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D", "E"])
df.plot(kind="box",vert=False)

```

[2]: <Axes: >

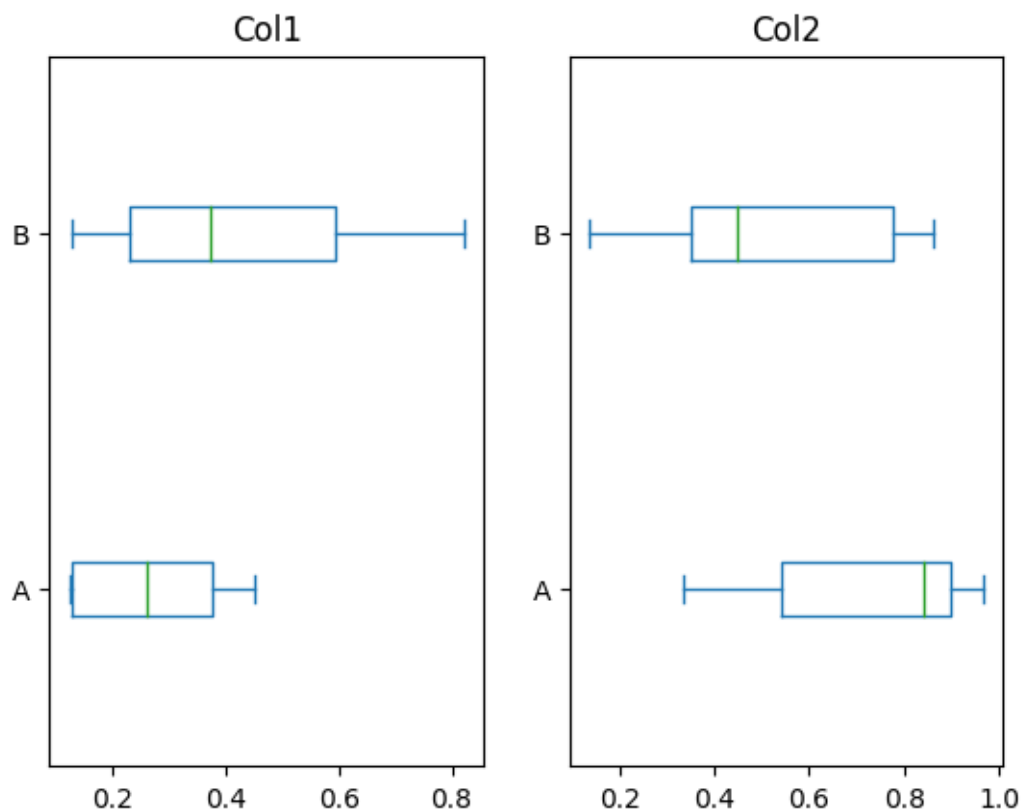


1.0.74 绘制数据的分组箱线图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(10, 2), columns=["Col1", "Col2"])
# 添加一行字符串，可以作为分类变量
df["X"] = pd.Series(["A", "A", "A", "A", "A", "B", "B", "B", "B", "B"])
df.plot(kind="box", vert=False, by="X")

[2]: Col1      Axes(0.125,0.11;0.352273x0.77)
Col2      Axes(0.547727,0.11;0.352273x0.77)
dtype: object
```



1.0.75 绘制数据的分组箱线图，多个分组变量

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

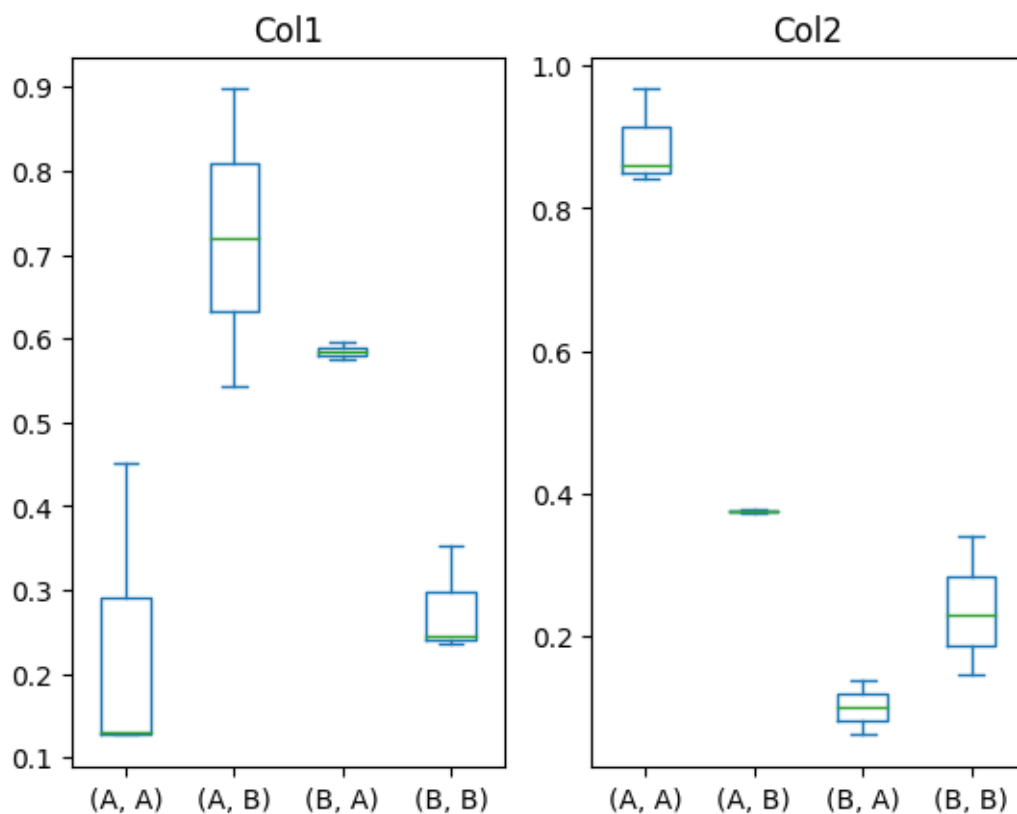
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(10, 3), columns=["Col1", "Col2", "Col3"])
# 添加一列
df["X"] = pd.Series(["A", "A", "A", "A", "A", "B", "B", "B", "B", "B"])
# 添加一列
df["Y"] = pd.Series(["A", "B", "A", "B", "A", "B", "A", "B", "A", "B"])
df.head()
```

```
[2]:      Col1      Col2      Col3  X  Y
0  0.126970  0.966718  0.260476  A  A
1  0.897237  0.376750  0.336222  A  B
2  0.451376  0.840255  0.123102  A  A
3  0.543026  0.373012  0.447997  A  B
```

```
4 0.129441 0.859879 0.820388 A A
```

```
[3]: df.plot(kind="box", column=["Col1", "Col2"], by=["X", "Y"])
```

```
[3]: Col1      Axes(0.125,0.11;0.352273x0.77)
Col2      Axes(0.547727,0.11;0.352273x0.77)
dtype: object
```

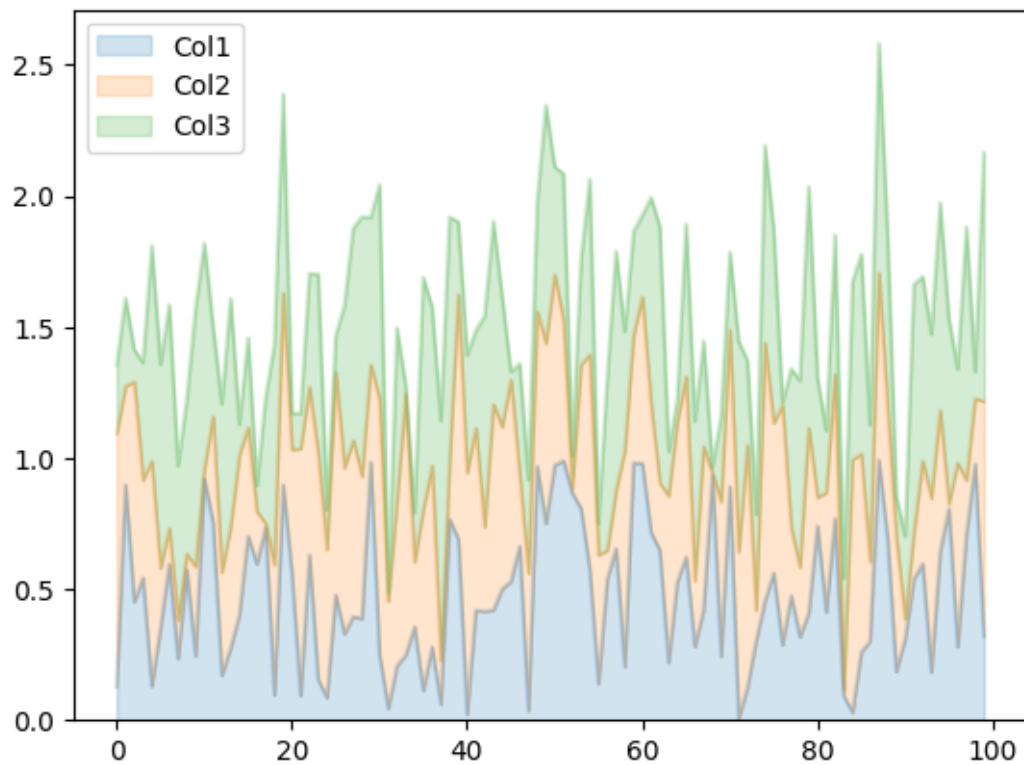


1.0.76 绘制多列数据的面积填充图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(100, 3), columns=["Col1", "Col2", "Col3"])
df.plot(kind="area", alpha=0.2)
```

```
[2]: <Axes: >
```

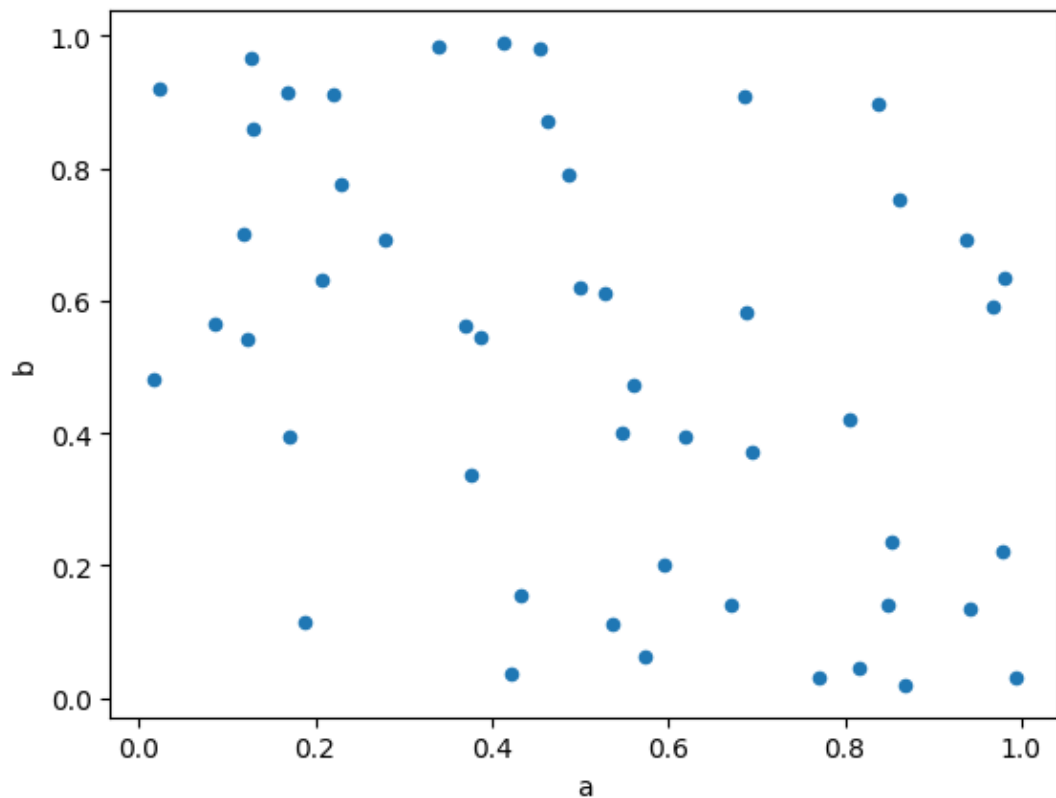


1.0.77 绘制两列数据的散点图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
# 新增一列
df["species"] = pd.Categorical(
    ["setosa"] * 20 + ["versicolor"] * 20 + ["virginica"] * 10
)
df.plot.scatter(x="a", y="b")

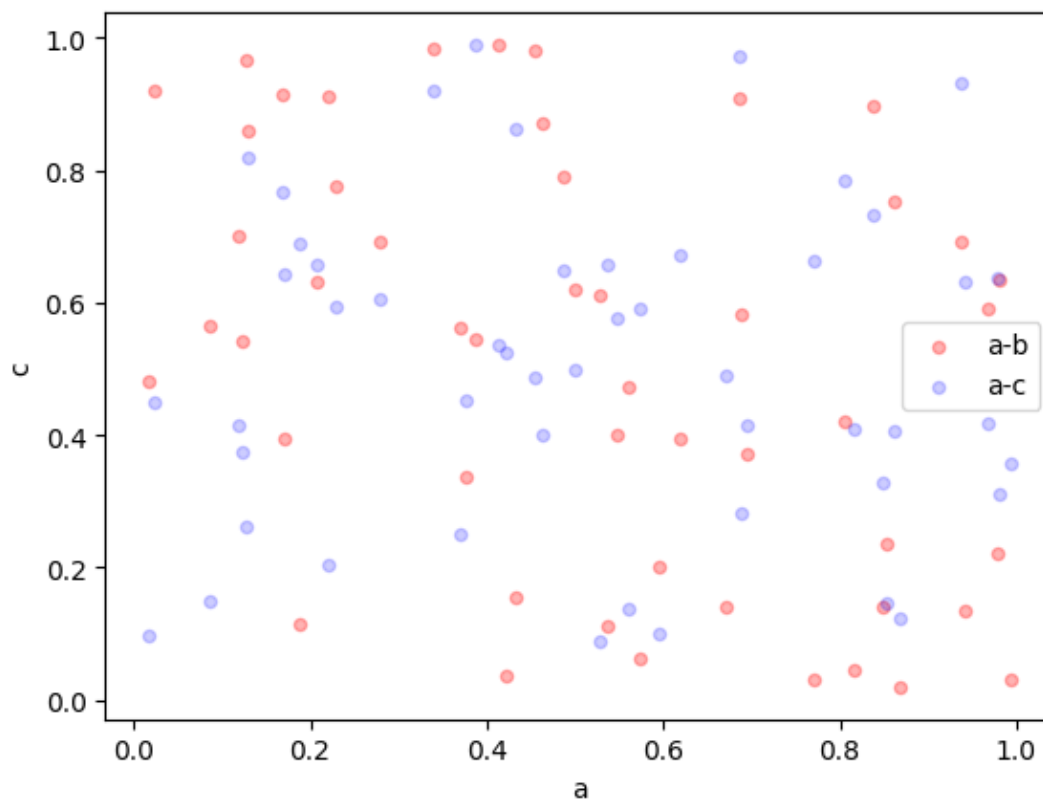
[2]: <Axes: xlabel='a', ylabel='b'>
```



1.0.78 绘制多对散点关系的数据到同一张图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
# 新增一列
df["species"] = pd.Categorical(
    ["setosa"] * 20 + ["versicolor"] * 20 + ["virginica"] * 10
)
ax = df.plot.scatter(x="a", y="b", color="red", alpha=0.3, label="a-b")
ax = df.plot.scatter(x="a", y="c", ax=ax, color="blue", alpha=0.2, label="a-c")
```

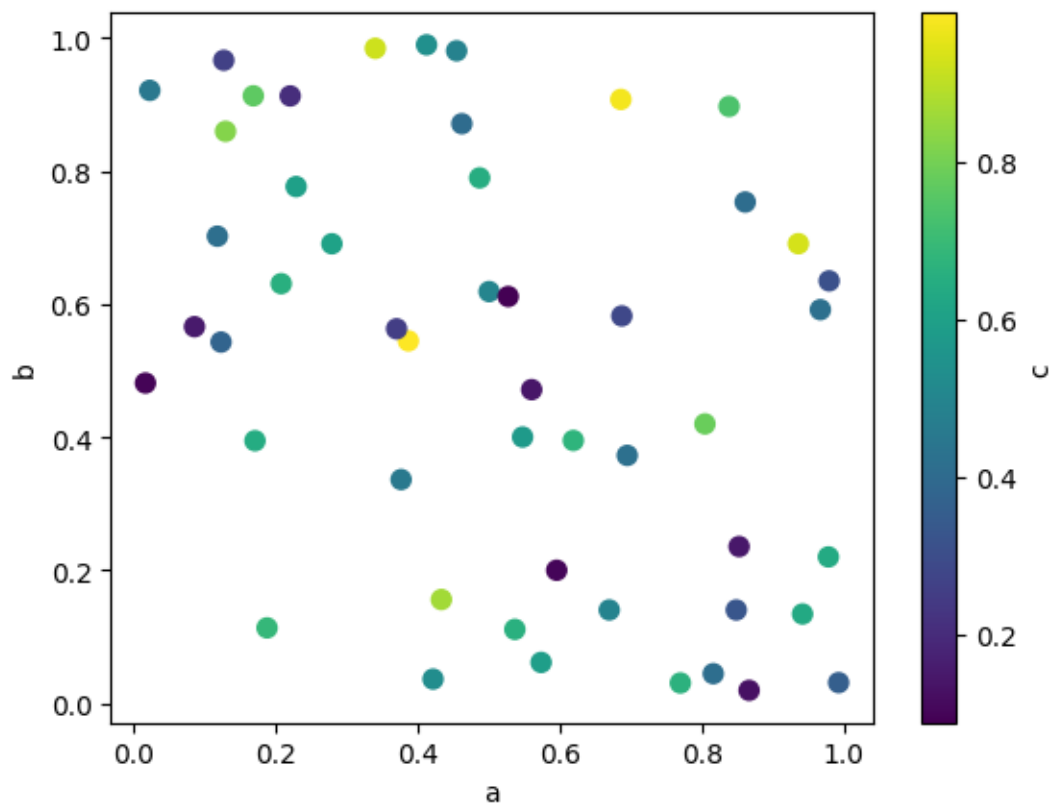


1.0.79 绘制数据的散点图，将颜色映射到另一个数值变量上

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
# 新增一列
df["species"] = pd.Categorical(
    ["setosa"] * 20 + ["versicolor"] * 20 + ["virginica"] * 10
)
df.plot.scatter(x="a", y="b", c="c", s=50)

[2]: <Axes: xlabel='a', ylabel='b'>
```

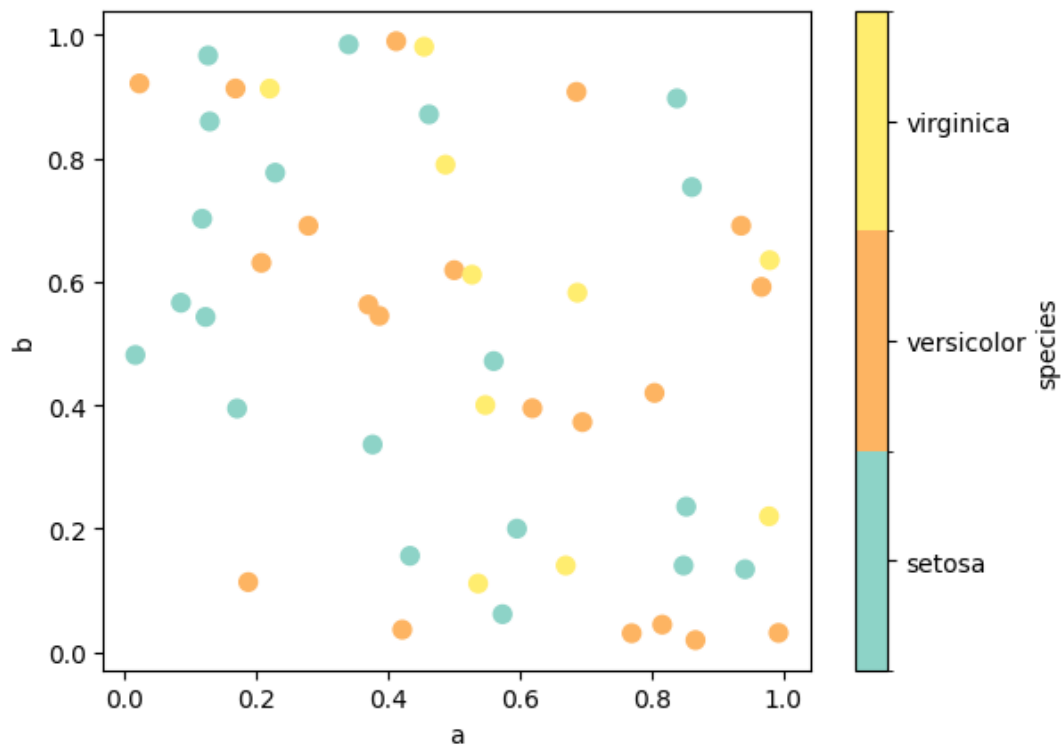


1.0.80 绘制数据的散点图，将颜色映射到另一个分类变量上

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
# 新增一列
df["species"] = pd.Categorical(
    ["setosa"] * 20 + ["versicolor"] * 20 + ["virginica"] * 10
)
df.plot.scatter(x="a", y="b", c="species", cmap="Set3", s=50)
```

[2]: <Axes: xlabel='a', ylabel='b'>

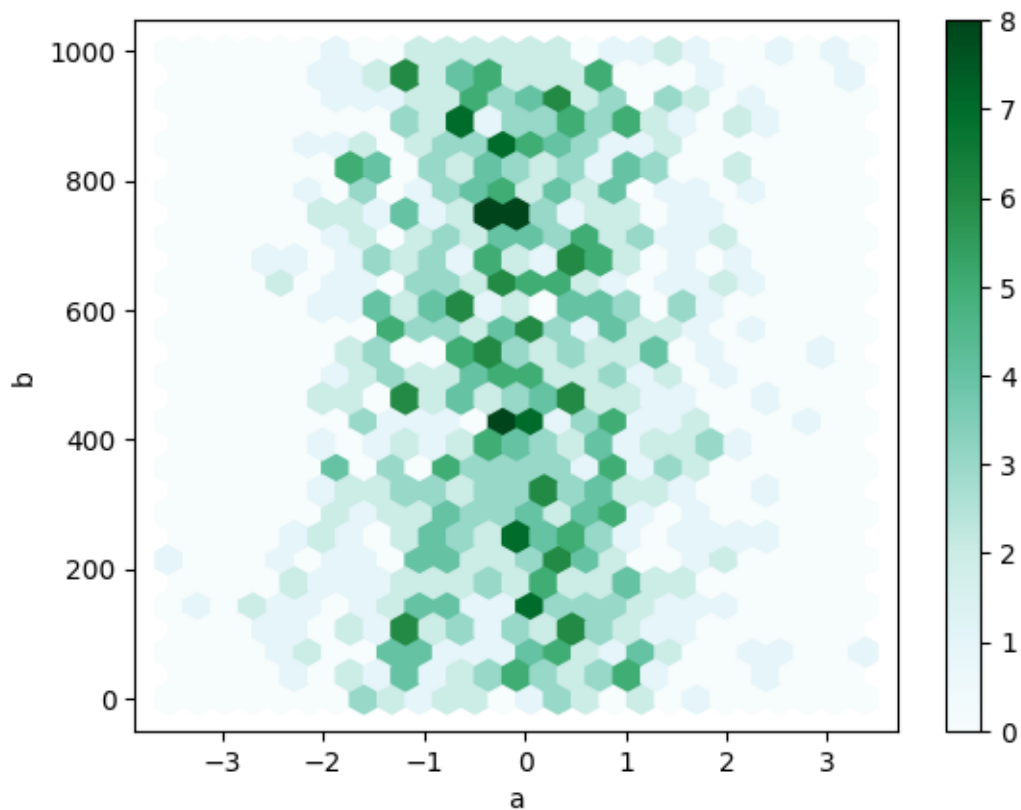


1.0.81 绘制数据的六边形箱图，针对散点过于密集的情况

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(np.random.randn(1000, 2), columns=["a", "b"])
# 修改这一列的值
df["b"] = df["b"] + np.arange(1000)
df.plot.hexbin(
    x="a", y="b",
    gridsize=25 # 六边形的大小，数值越大，面积越小
)

[2]: <Axes: xlabel='a', ylabel='b'>
```

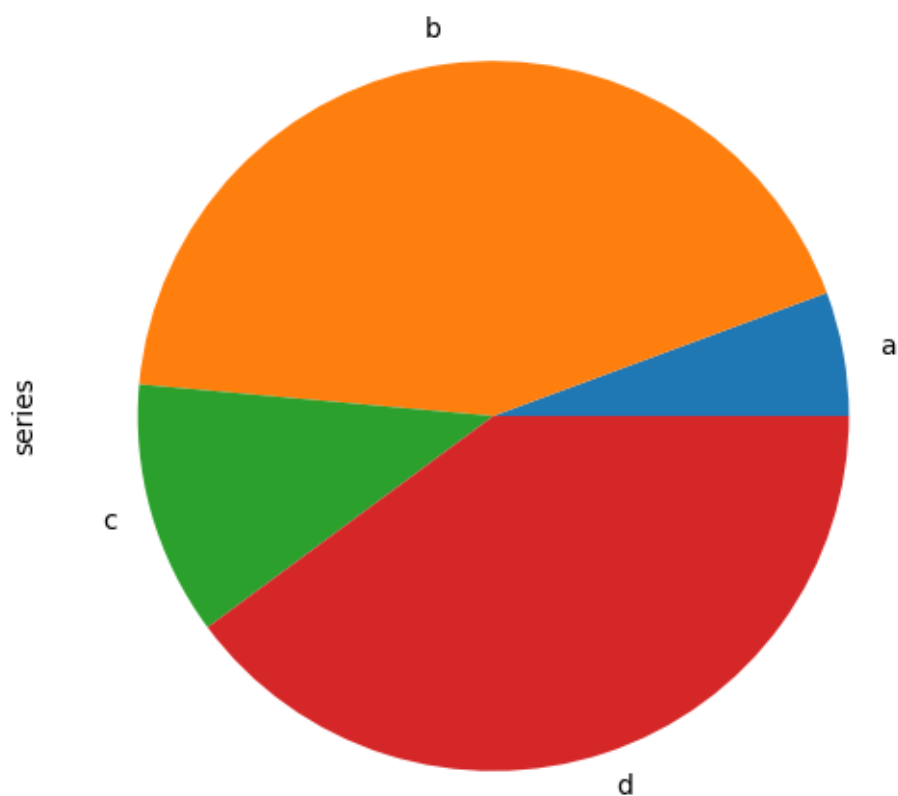



1.0.82 绘制饼图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
series = pd.Series(3 * np.random.rand(4), index=["a", "b", "c", "d"], name="series")
series.plot.pie(figsize=(6, 6))

[2]: <Axes: ylabel='series'>
```

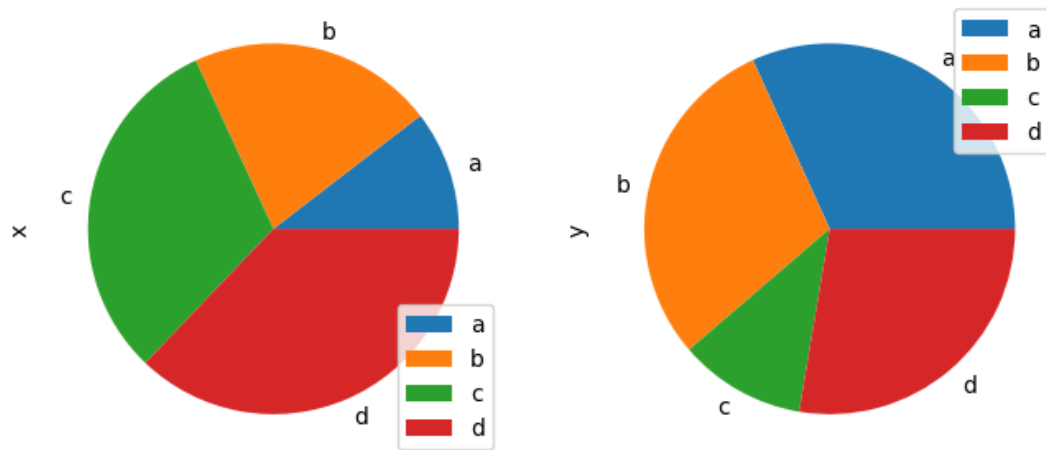


1.0.83 绘制多列数据的饼图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 设置随机数种子
np.random.seed(123456)
# 生成数据
df = pd.DataFrame(
    3 * np.random.rand(4, 2), index=["a", "b", "c", "d"], columns=["x", "y"]
)
# 以子图形式
df.plot.pie(subplots=True, figsize=(8, 4))

[2]: array([<Axes: ylabel='x'>, <Axes: ylabel='y'>], dtype=object)
```



1.0.84 各类绘图对缺失值的处理方式如下

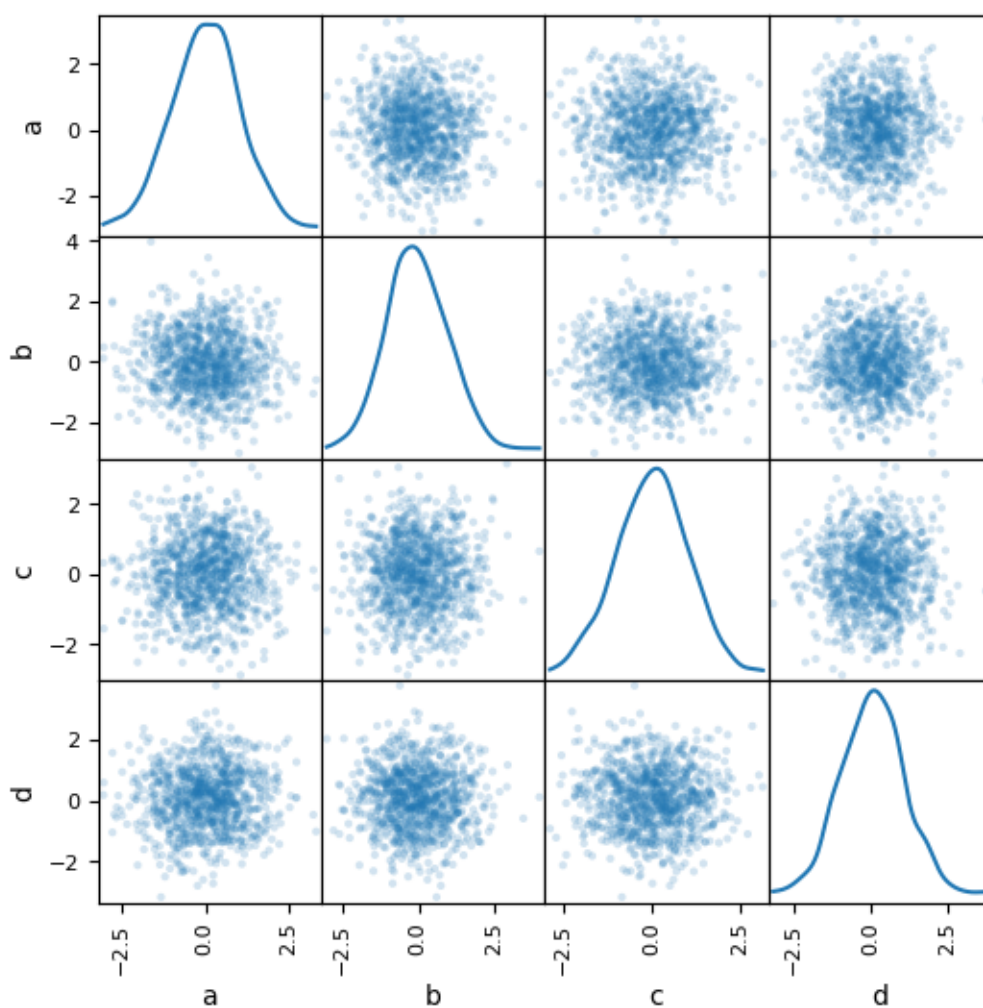
Plot Type	NaN Handling
Line	Leave gaps at NaNs
Line	(stacked) Fill 0's
Bar	Fill 0's
Scatter	Drop NaNs
Histogram	Drop NaNs (column-wise)
Box	Drop NaNs (column-wise)
Area	Fill 0's
KDE	Drop NaNs (column-wise)
Hexbin	Drop NaNs
Pie	Fill 0's

1.0.85 绘制多个变量之间的散点图矩阵

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd

[2]: # 导入散点矩阵绘图工具
from pandas.plotting import scatter_matrix
# 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(np.random.randn(1000, 4), columns=["a", "b", "c", "d"])
scatter_matrix(df, alpha=0.2, figsize=(6, 6), diagonal="kde")
```

```
[2]: array([[<Axes: xlabel='a', ylabel='a'>, <Axes: xlabel='b', ylabel='a'>,
          <Axes: xlabel='c', ylabel='a'>, <Axes: xlabel='d', ylabel='a'>],
          [<Axes: xlabel='a', ylabel='b'>, <Axes: xlabel='b', ylabel='b'>,
          <Axes: xlabel='c', ylabel='b'>, <Axes: xlabel='d', ylabel='b'>],
          [<Axes: xlabel='a', ylabel='c'>, <Axes: xlabel='b', ylabel='c'>,
          <Axes: xlabel='c', ylabel='c'>, <Axes: xlabel='d', ylabel='c'>],
          [<Axes: xlabel='a', ylabel='d'>, <Axes: xlabel='b', ylabel='d'>,
          <Axes: xlabel='c', ylabel='d'>, <Axes: xlabel='d', ylabel='d'>]],
      dtype=object)
```

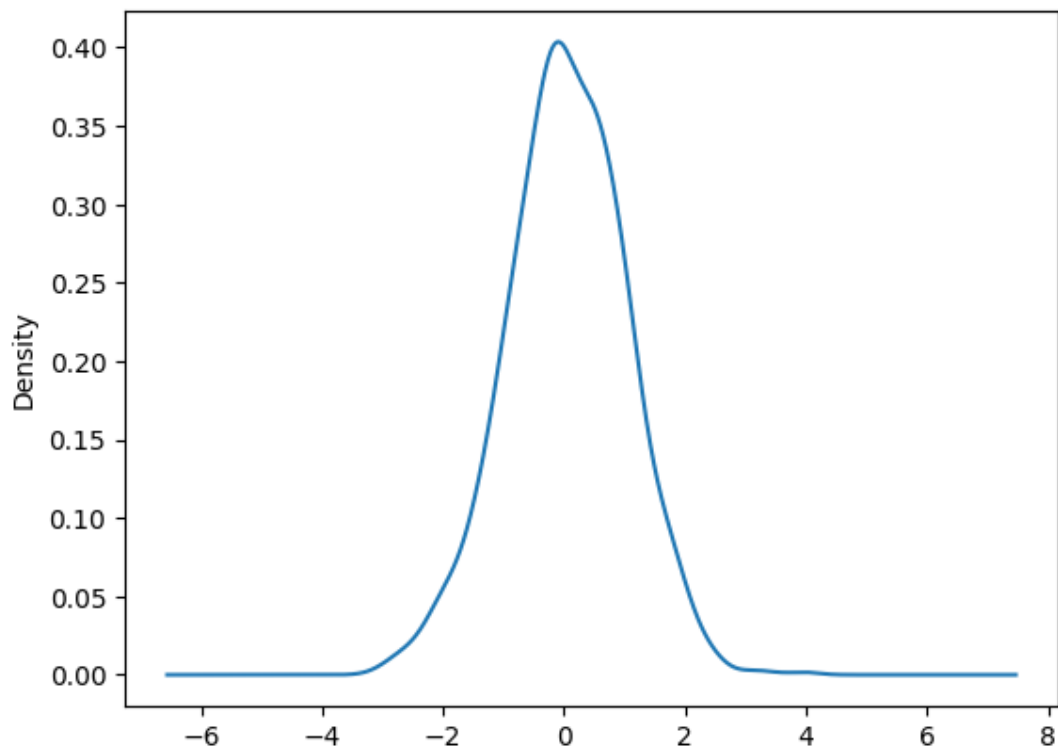


1.0.86 绘制核密度估计曲线

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 设置随机数种子
np.random.seed(1)
ser = pd.Series(np.random.randn(1000))
ser.plot.kde()
```

```
[2]: <Axes: ylabel='Density'>
```



1.0.87 绘制安德鲁曲线，多元分析中的一种图形

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入数据集获取工具
from sklearn.datasets import fetch_openml
```

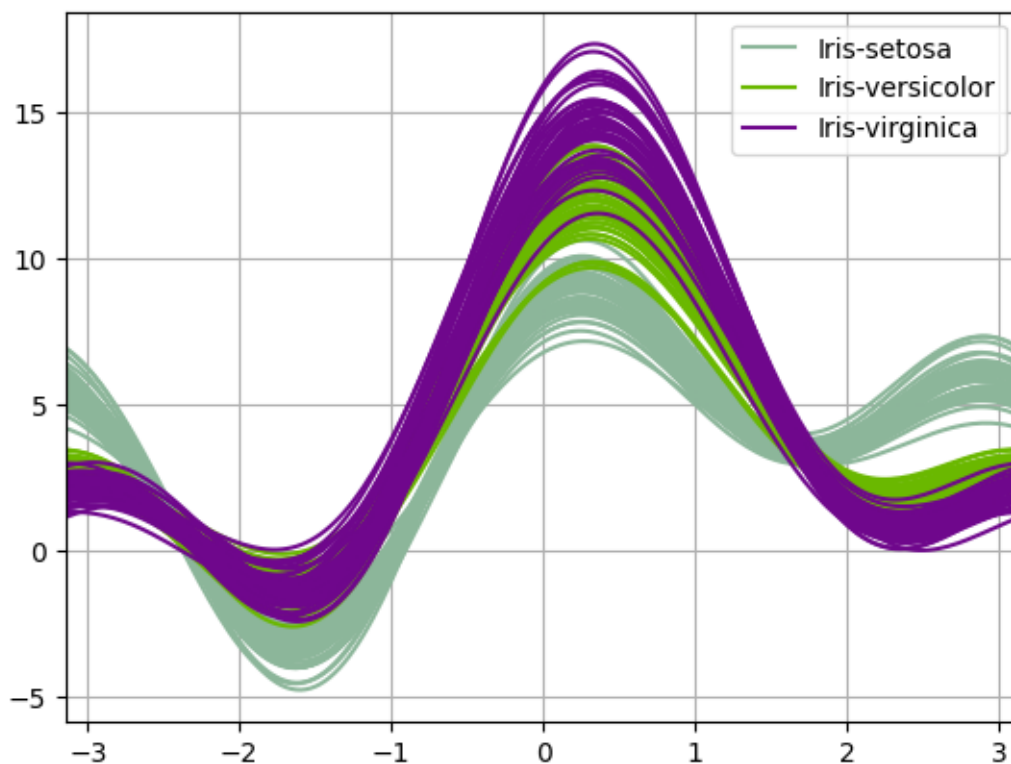
```
[2]: # 导入安德鲁曲线绘图工具
from pandas.plotting import andrews_curves
# 获取数据
data = fetch_openml(
    data_id=61,
    as_frame=True,
    parser="pandas"
)["frame"]
```

```
data.head()
```

```
[2]:   sepalength  sepalwidth  petallength  petalwidth    class
0         5.1         3.5         1.4         0.2  Iris-setosa
1         4.9         3.0         1.4         0.2  Iris-setosa
2         4.7         3.2         1.3         0.2  Iris-setosa
3         4.6         3.1         1.5         0.2  Iris-setosa
4         5.0         3.6         1.4         0.2  Iris-setosa
```

```
[3]: andrews_curves(data, "class")
```

```
[3]: <Axes: >
```



1.0.88 绘制平行坐标曲线，多元分析中的一种图形

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入数据集获取工具
from sklearn.datasets import fetch_openml
```

```
[2]: # 导入安德鲁曲线绘图工具
from pandas.plotting import parallel_coordinates
```

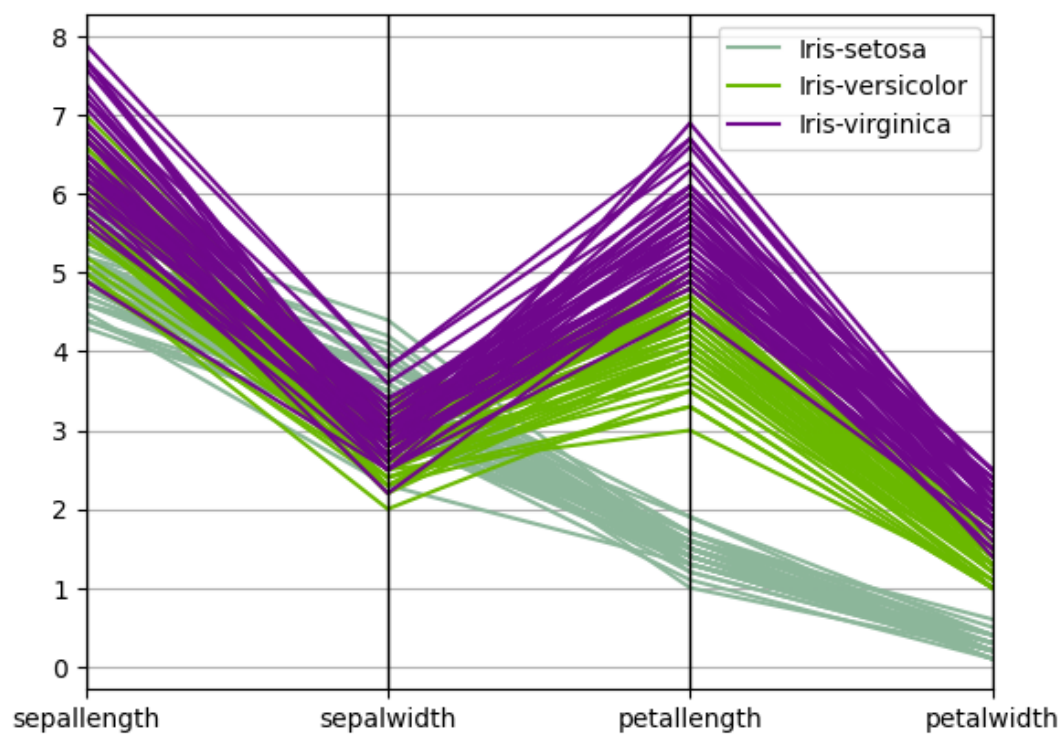
```
# 获取数据
data = fetch_openml(
    data_id=61,
    as_frame=True,
    parser="pandas"
)["frame"]
data.head()
```

```
[2]:
```

	sepalength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
[3]: parallel_coordinates(data, "class")
```

```
[3]: <Axes: >
```

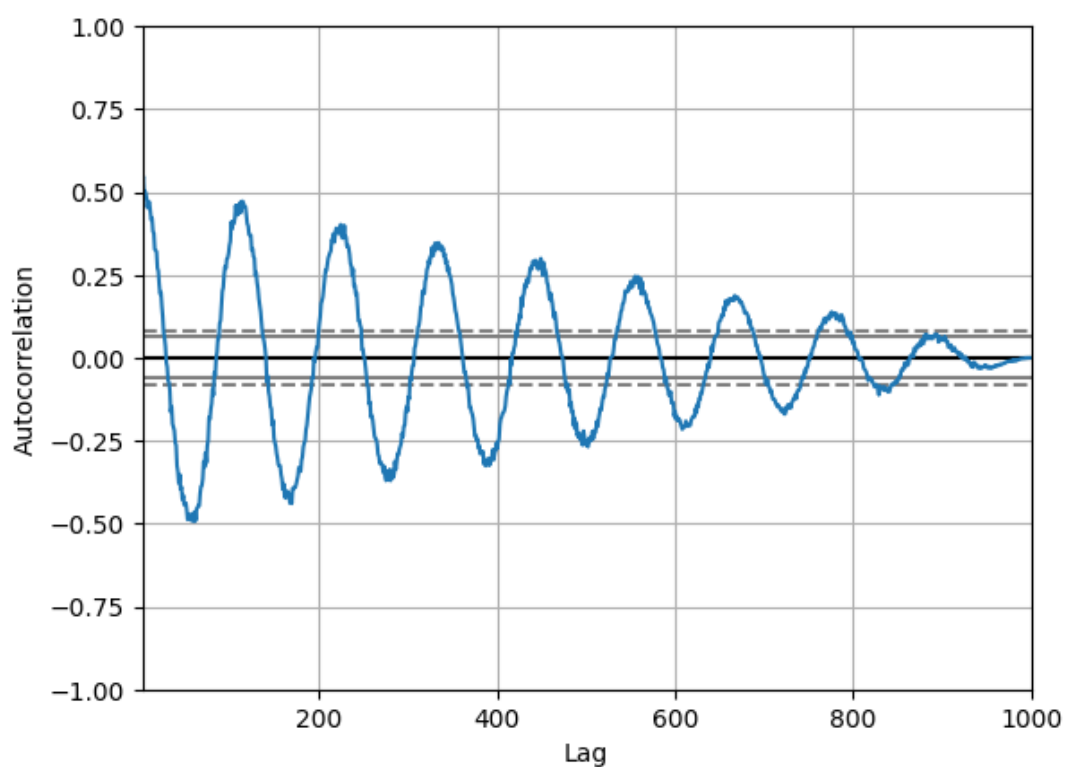


1.0.89 绘制自相关图

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
```

```
[2]: # 导入自相关系数绘图工具
from pandas.plotting import autocorrelation_plot
spacing = np.linspace(-9 * np.pi, 9 * np.pi, num=1000)
# 设置随机数种子
np.random.seed(1)
# 生成数据
data = pd.Series(0.7 * np.random.rand(1000) + 0.3 * np.sin(spacing))
autocorrelation_plot(data)
```

```
[2]: <Axes: xlabel='Lag', ylabel='Autocorrelation'>
```



1.0.90 调整数据的显示精度

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 生成数据
df = pd.DataFrame({
    "strings": ["Adam", "Mike"],
    "ints": [1, 3],
```



```

        "floats": [1.123, 1000.23]
    })
df.head()

```

```

[2]:   strings  ints  floats
0    Adam    1    1.123
1    Mike    3  1000.230

```

```

[3]: # 格式化输出
df.style.format(
    precision=3, thousands=".", decimal="," # 将千分位的点替换为,
).format_index(
    str.upper, axis=1 # 将列名全部大写
).relabel_index(
    ["row 1", "row 2"], axis=0 # 修改行名
)

```

```

[3]: <pandas.io.formats.style.Styler at 0x20649a5e5f0>

```

1.0.91 自定义高亮显示数据

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
weather_df = pd.DataFrame(
    np.random.rand(10,2)*5,
    index=pd.date_range(start="2021-01-01", periods=10),
    columns=["Tokyo", "Beijing"]
)
# 自定义函数
def rain_condition(v):
    if v < 1.75:
        return "Dry"
    elif v < 2.75:
        return "Rain"
    return "Heavy Rain"
# 高亮显示
def make_pretty(styler):
    styler.set_caption("Weather Conditions") # 设置数据表的标题

```

```

styler.format(rain_condition) # 设置数据格式, 调用自定义函数
styler.format_index(lambda v: v.strftime("%A")) # 修改行名
# 背景颜色
styler.background_gradient(axis=None, vmin=1, vmax=5, cmap="YlGnBu")
return styler

weather_df

```

```

[2]:           Tokyo    Beijing
2021-01-01  2.085110  3.601622
2021-01-02  0.000572  1.511663
2021-01-03  0.733779  0.461693
2021-01-04  0.931301  1.727804
2021-01-05  1.983837  2.694084
2021-01-06  2.095973  3.426098
2021-01-07  1.022261  4.390587
2021-01-08  0.136938  3.352338
2021-01-09  2.086524  2.793449
2021-01-10  0.701935  0.990507

```

```

[3]: # 高亮显示, 调用 make_pretty 函数
weather_df.loc["2021-01-04":"2021-01-08"].style.pipe(make_pretty)

```

```

[3]: <pandas.io.formats.style.Styler at 0x287e35be0e0>

```

1.0.92 隐藏行名和列名

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(np.random.randn(5, 5))
df

```

```

[2]:           0          1          2          3          4
0  1.624345 -0.611756 -0.528172 -1.072969  0.865408
1 -2.301539  1.744812 -0.761207  0.319039 -0.249370
2  1.462108 -2.060141 -0.322417 -0.384054  1.133769
3 -1.099891 -0.172428 -0.877858  0.042214  0.582815
4 -1.100619  1.144724  0.901591  0.502494  0.900856

```

```
[3]: df.style.hide(
        subset=[0, 2, 4], axis=0 # 隐藏第一, 三, 五行
    ).hide(
        subset=[0, 2, 4], axis=1 # 隐藏第一, 三, 五列
    )
```

```
[3]: <pandas.io.formats.style.Styler at 0x241561cfac0>
```

```
[4]: # 需要显示的行列
show = [0, 2, 4]
df.style.hide(
    [row for row in df.index if row not in show], axis=0 # 隐藏 show 列表之外的行
).hide(
    [col for col in df.columns if col not in show], axis=1 # 隐藏 show 列表之外的列
)
```

```
[4]: <pandas.io.formats.style.Styler at 0x2415f52fd30>
```

1.0.93 连接数据

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(np.random.randn(5, 5))
df
```

```
[2]:
```

	0	1	2	3	4
0	1.624345	-0.611756	-0.528172	-1.072969	0.865408
1	-2.301539	1.744812	-0.761207	0.319039	-0.249370
2	1.462108	-2.060141	-0.322417	-0.384054	1.133769
3	-1.099891	-0.172428	-0.877858	0.042214	0.582815
4	-1.100619	1.144724	0.901591	0.502494	0.900856

```
[3]: summary_styler = df.agg(["sum", "mean"]).style.format(
        precision=3
    ).relabel_index(["Sum", "Average"]) # 中心修改行名
summary_styler
```

```
[3]: <pandas.io.formats.style.Styler at 0x1caeafda50>
```

```
[4]: df.style.format(precision=1).concat(summary_styler)
```

[4]: <pandas.io.formats.style.Styler at 0x1caed690bb0>

1.0.94 将数据单元格进行格式转换

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x22964ffda50>

```
[3]: # 对负数进行格式转换
def style_negative(v, props=''):
    return props if v < 0 else None
s = df.style.applymap(
    style_negative, props="color:red;"
).applymap(
    lambda v: "opacity: 20%;" if (v < 0.3) and (v > -0.3) else None
)
s
```

[3]: <pandas.io.formats.style.Styler at 0x2295ec329b0>

1.0.95 对每一列的最大值进行高亮显示

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x145f5ac1e10>

```
[3]: # 对最大值进行高亮显示
def highlight_max(s, props=""):
    return np.where(s == np.nanmax(s.values), props, "")
df.style.apply(
    highlight_max, props="color:white;background-color:darkblue",
    axis=0
)
```

```
[3]: <pandas.io.formats.style.Styler at 0x145ef7b0670>
```

1.0.96 对每一行以及整个 **dataframe** 的最大值进行高亮显示

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x14caa885ea0>
```

```
[3]: # 对最大值进行高亮显示
def highlight_max(s, props=""):
    return np.where(s == np.nanmax(s.values), props, "")
```

```
[4]: df.style.apply(
    highlight_max, props="color:white;background-color:pink;",
    axis=1
)
```

```
[4]: <pandas.io.formats.style.Styler at 0x14ca452c310>
```

```
[5]: df.style.apply(
    highlight_max, props="color:white;background-color:purple",
    axis=None
)
```

```
[5]: <pandas.io.formats.style.Styler at 0x14cad00ceb0>
```

1.0.97 对行名和列名进行格式变换

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style

[2]: <pandas.io.formats.style.Styler at 0x2461ce5da50>

[3]: df.style.applymap_index(
    lambda v: "color: pink;" if v>4 else "color: darkblue;",
    axis=0 # 对行名进行变换
)

[3]: <pandas.io.formats.style.Styler at 0x24616a92980>

[4]: df.style.apply_index(
    lambda s: np.where(
        s.isin(["A", "B"]), "color: pink;", "color: darkblue;"
    ),
    axis=1 # 对列名进行转换
)

[4]: <pandas.io.formats.style.Styler at 0x24616a93b20>
```

1.0.98 对行名和列名指定子集进行格式变换

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x19a12e95e10>

```
[3]: # 对最大值进行高亮显示
def highlight_max(s, props=""):
    return np.where(s == np.nanmax(s.values), props, "")

slice_ = ["C", "D"]
df.style.apply(
    highlight_max, props="color:red;",
    axis=0, subset=slice_
).set_properties(
    **{"background-color": "#ffffb3"},
    subset=slice_
)
```

[3]: <pandas.io.formats.style.Styler at 0x19a15e839d0>

1.0.99 对缺失值进行高亮显示（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x1763425da50>

```
[3]: df.iloc[0,2] = np.nan
df.iloc[4,3] = np.nan
```

```
[4]: df.style.highlight_null(color="yellow")
```

[4]: <pandas.io.formats.style.Styler at 0x176369e8d90>

1.0.100 对最大值和最小值进行高亮显示（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
```

```
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x26e697b5ea0>
```

```
[3]: # 最大值高亮显示
df.style.highlight_max(
    axis=1, # 行最大值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[3]: <pandas.io.formats.style.Styler at 0x26e6bf3cd60>
```

```
[4]: # 最大值高亮显示
df.style.highlight_max(
    axis=0, # 列最大值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[4]: <pandas.io.formats.style.Styler at 0x26e6bf3d270>
```

```
[5]: # 最大值高亮显示
df.style.highlight_max(
    axis=None, # 整个 dataframe 最大值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[5]: <pandas.io.formats.style.Styler at 0x26e6c79f910>
```

```
[6]: # 最小值高亮显示
df.style.highlight_min(
    axis=1, # 行最小值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[6]: <pandas.io.formats.style.Styler at 0x26e6c79f3a0>
```

```
[7]: # 最小值高亮显示
df.style.highlight_min(
    axis=0, # 列最小值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[7]: <pandas.io.formats.style.Styler at 0x26e6c79fc40>
```



```
[8]: # 最小值高亮显示
df.style.highlight_min(
    axis=None, # 整个 dataframe 最小值
    props="color:white; font-weight:bold; background-color:darkblue;"
)
```

```
[8]: <pandas.io.formats.style.Styler at 0x26e6c79f6d0>
```

1.0.101 对介于某两个数值之间的数据进行高亮显示（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A", "B", "C", "D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x24b751a5ea0>
```

```
[3]: left = pd.Series([1.0, 0.0, 1.0], index=["A", "B", "D"])
df.style.highlight_between(
    left=left, right=1.5,
    axis=1, # 对列进行操作
    props="color:white; background-color:purple;"
)
```

```
[3]: <pandas.io.formats.style.Styler at 0x24b7792cfd0>
```

```
[4]: left = pd.Series([0, 0, 0, 0, 0], index=[0,2,3,5,8])
df.style.highlight_between(
    left=left, right=2.5,
    axis=0, # 对行进行操作
    props="color:white; background-color:purple;"
)
```

```
[4]: <pandas.io.formats.style.Styler at 0x24b6ee4fa60>
```

1.0.102 对分位数进行高亮显示（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x1b9f30f9ea0>
```

```
[3]: # 将 0.85 到 1 的分位数高亮显示
df.style.highlight_quantile(
    q_left=0.85, axis=0, color="yellow" # 列
)
```

```
[3]: <pandas.io.formats.style.Styler at 0x1b9f587cd60>
```

```
[4]: df.style.highlight_quantile(
    q_left=0.85, axis=1, color="yellow" # 行
)
```

```
[4]: <pandas.io.formats.style.Styler at 0x1b9f587d4e0>
```

```
[5]: df.style.highlight_quantile(
    q_left=0.85, axis=None, color="yellow" # 整个 dataframe
)
```

```
[5]: <pandas.io.formats.style.Styler at 0x1b9f60df580>
```

1.0.103 对 dataframe 添加背景色（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
```

```
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x21c7ec7da50>
```

```
[3]: import seaborn as sns
cm = sns.light_palette("green", as_cmap=True)
df.style.background_gradient(cmap=cm)
```

```
[3]: <pandas.io.formats.style.Styler at 0x21c788b0bb0>
```

1.0.104 对 dataframe 添加文字前景色（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x293c5b7da50>
```

```
[3]: import seaborn as sns
cm = sns.light_palette("green", as_cmap=True)
df.style.text_gradient(cmap=cm)
```

```
[3]: <pandas.io.formats.style.Styler at 0x293bf82f850>
```

1.0.105 对整个 dataframe 设置背景色和前景色（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x231dabd5e10>

```
[3]: # 对 style 设置属性
df.style.set_properties(
    **{
        "background-color": "black",
        "color": "lawngreen",
        "border-color": "white"
    }
)
```

[3]: <pandas.io.formats.style.Styler at 0x231dd36d210>

1.0.106 在 dataframe 上添加柱状图（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A", "B", "C", "D"])
df.style
```

[2]: <pandas.io.formats.style.Styler at 0x1711ec3da50>

```
[3]: df.style.bar(subset=["A", "D"], color='#d65f5f')
```

[3]: <pandas.io.formats.style.Styler at 0x171213d0ca0>

1.0.107 在 dataframe 上添加柱状图，不遮挡数值（内置函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A", "B", "C", "D"])
```

```
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x20dad88da50>
```

```
[3]: df.style.format(
      "{:.3f}", na_rep=""
    ).bar(
      align=0, vmin=-2.5, vmax=2.5, cmap="bwr", height=50,
      width=60, props="width: 120px; border-right: 1px solid black;"
    ).text_gradient(cmap="bwr", vmin=-2.5, vmax=2.5)
```

```
[3]: <pandas.io.formats.style.Styler at 0x20da74c6980>
```

1.0.108 在 dataframe 上使用鼠标互动

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(10,4), columns=["A","B","C","D"])
df.style
```

```
[2]: <pandas.io.formats.style.Styler at 0x204dabc1ea0>
```

```
[3]: # 鼠标停留时，行高和列宽增大的函数
def magnify():
    return [dict(selector="th",
                  props=[("font-size", "10pt")]),
            dict(selector="td",
                  props=[('padding', "8em 8em")]),
            dict(selector="th:hover",
                  props=[("font-size", "10pt")]),
            dict(selector="tr:hover td:hover",
                  props=[('max-width', '5px'),
                          ('font-size', '5pt')])]
]
```

```
[4]: np.random.seed(25)
import seaborn as sns
cmap = sns.diverging_palette(5, 250, as_cmap=True)
df.style.background_gradient(
```

```

        cmap, axis=1
    ).set_properties(
        **{
            "max-width": "80px", "font-size": "1pt"
        }
    ).set_caption(
        "Hover to magnify"
    ).format(
        precision=2
    ).set_table_styles(magnify())

```

[4]: <pandas.io.formats.style.Styler at 0x204ddbaf880>

1.0.109 固定 dataframe 的行名（列名不需要固定）

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

[2]: # 设置随机数种子
np.random.seed(0)
# 生成数据
df = pd.DataFrame(np.random.randn(16, 100), columns=["A{}".format(i) for i in
↪range(1, 101)])
df.style

```

[2]: <pandas.io.formats.style.Styler at 0x14fdbbbede10>

```

[3]: # 固定行名
df.style.set_sticky(axis="index")

```

[3]: <pandas.io.formats.style.Styler at 0x14fdebdf2e0>

1.0.110 分组求和

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

[2]: df = pd.DataFrame({"X": ["B", "B", "A", "A"], "Y": [1, 2, 3, 4]})
df

```

```
[2]:      X  Y
      0  B  1
      1  B  2
      2  A  3
      3  A  4
```

```
[3]: # 按照变量 X 来分组，对剩下的变量求和
      df.groupby(["X"]).sum()
```

```
[3]:      Y
      X
      A  7
      B  3
```

```
[4]: # 分组求和后，排序
      df.groupby(["X"], sort=False).sum()
```

```
[4]:      Y
      X
      B  3
      A  7
```

1.0.111 分组后查看分组数据

```
[1]: # 导入基础计算库
      import numpy as np
      # 导入数据分析库
      import pandas as pd
      # 导入 matplotlib 包
      import matplotlib as mpl
```

```
[2]: df = pd.DataFrame({"X": ["B", "B", "A", "A"], "Y": [1, 2, 3, 4]})
      df
```

```
[2]:      X  Y
      0  B  1
      1  B  2
      2  A  3
      3  A  4
```

```
[3]: # 获取 A 组别
      df.groupby(["X"]).get_group("A")
```

```
[3]:      X  Y
      2  A  3
      3  A  4
```

```
[4]: # 获取 B 组别
df.groupby(["X"]).get_group("B")
```

```
[4]:    X  Y
0   B  1
1   B  2
```

1.0.112 按照行名分组，针对多水平索引的情况

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

[2]: df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:    A      B      C      D
0  foo   one -0.827937 -1.798664
1  bar   one  0.762386 -0.411717
2  foo   two -0.313361  0.071818
3  bar three  1.199653  1.508659
4  foo   two  0.919133  0.012376
5  bar   two -0.033925  1.043274
6  foo   one  1.581738 -0.798991
7  foo three  0.652731 -0.568396
```

```
[3]: # 重新设置索引，将索引设置为 A, B 列的值
df2 = df.set_index(["A", "B"])
df2
```

```
[3]:           C      D
A  B
foo one -0.827937 -1.798664
bar one  0.762386 -0.411717
foo two -0.313361  0.071818
bar three 1.199653  1.508659
foo two  0.919133  0.012376
```



```
bar two    -0.033925    1.043274
foo one     1.581738   -0.798991
      three    0.652731   -0.568396
```

```
[4]: # 按照 index 的 A 进行分组
grouped = df2.groupby(level=df2.index.names.difference(["B"]))
# 对分组后的数据求和
grouped.sum()
```

```
[4]:           C          D
A
bar  1.928113  2.140216
foo  2.012304 -3.081857
```

1.0.113 同时按照行名和列名分组，针对多水平索引的情况

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 生成数据
arrays = [
    ["bar", "bar", "baz", "baz", "foo", "foo", "qux", "qux"],
    ["one", "two", "one", "two", "one", "two", "one", "two"],
]
# 生成多水平索引
index = pd.MultiIndex.from_arrays(arrays, names=["first", "second"])
# 生成数据
df = pd.DataFrame({"A": [1, 1, 1, 1, 2, 2, 3, 3], "B": np.arange(8)}, index=index)
df
```

```
[2]:           A  B
first second
bar   one    1  0
      two    1  1
baz   one    1  2
      two    1  3
foo   one    2  4
      two    2  5
qux   one    3  6
      two    3  7
```

```
[3]: # 按照第二水平索引 (second) 和 A 列进行分组, 再求和
df.groupby([pd.Grouper(level=1), "A"]).sum()
```

```
[3]:          B
second A
one    1  2
       2  4
       3  6
two    1  4
       2  5
       3  7
```

1.0.114 迭代循环输出分组的组别名和分组数据

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(10)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.331587  0.004291
1  bar   one  0.715279 -0.174600
2  foo   two -1.545400  0.433026
3  bar  three -0.008384  1.203037
4  foo   two  0.621336 -0.965066
5  bar   two -0.720086  1.028274
6  foo   one  0.265512  0.228630
7  foo  three  0.108549  0.445138
```

```
[3]: # 分组
grouped = df.groupby("A")
```

```
# 迭代循环
for name, group in grouped:
    print(name)
    print(group)
```

```
bar
   A      B      C      D
1 bar  one  0.715279 -0.174600
3 bar three -0.008384  1.203037
5 bar  two -0.720086  1.028274
foo
   A      B      C      D
0 foo  one  1.331587  0.004291
2 foo  two -1.545400  0.433026
4 foo  two  0.621336 -0.965066
6 foo  one  0.265512  0.228630
7 foo three  0.108549  0.445138
```

1.0.115 分组求和

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:   A      B      C      D
0 foo  one  1.624345  0.319039
1 bar  one -0.611756 -0.249370
2 foo  two -0.528172  1.462108
3 bar three -1.072969 -2.060141
4 foo  two  0.865408 -0.322417
```

```

5 bar    two -2.301539 -0.384054
6 foo    one  1.744812  1.133769
7 foo    three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求和
grouped[["C", "D"]].aggregate("sum")

```

```

[3]:           C          D
A
bar -3.986264 -2.693565
foo  2.945186  1.492608

```

1.0.116 分组求平均数

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df

```

```

[2]:           A          B          C          D
0  foo    one  1.624345  0.319039
1  bar    one -0.611756 -0.249370
2  foo    two -0.528172  1.462108
3  bar    three -1.072969 -2.060141
4  foo    two  0.865408 -0.322417
5  bar    two -2.301539 -0.384054
6  foo    one  1.744812  1.133769
7  foo    three -0.761207 -1.099891

```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求平均数
grouped[["C", "D"]].aggregate("mean")
```

```
[3]:           C           D
A
bar -1.328755 -0.897855
foo  0.589037  0.298522
```

1.0.117 分组计数

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:           A           B           C           D
0  foo     one  1.624345  0.319039
1  bar     one -0.611756 -0.249370
2  foo     two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo     two  0.865408 -0.322417
5  bar     two -2.301539 -0.384054
6  foo     one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求平均数
grouped[["C", "D"]].aggregate("count")
```

```
[3]:      C  D
      A
      bar  3  3
      foo  5  5
```

1.0.118 分组计算剩下的组别之间的协方差

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求平均数
grouped[["C", "D"]].aggregate("cov")
```

```
[3]:      C      D
      A
      bar C  0.762911 -0.166076
      D -0.166076  1.017716
```

```
foo C  1.388843  0.350805
      D  0.350805  1.098279
```

1.0.119 分组计算每一列的最大值的下标

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("idxmax")
```

```
[3]:      C  D
A
bar  1  1
foo  6  2
```

1.0.120 分组计算每一列的最小值的下标

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("idxmin")
```

```
[3]:      C  D
A
bar  5  3
foo  7  7
```


1.0.121 分组计算每一列的最大值

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("max")
```

```
[3]:
```

	C	D
A		
bar	-0.611756	-0.249370
foo	1.744812	1.462108

1.0.122 分组计算每一列的最小值

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("min")
```

```
[3]:      C      D
A
bar -2.301539 -2.060141
foo -0.761207 -1.099891
```

1.0.123 分组计算每一列的中位数

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("median")
```

```
[3]:
```

	C	D
A		
bar	-1.072969	-0.384054
foo	0.865408	0.319039

1.0.124 分组计算每一列中唯一值的数量

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("nunique")
```

```
[3]:      C  D
A
bar  3  3
foo  5  5
```

1.0.125 分组计算每一列的连乘

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo three -0.761207 -1.099891
```

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("prod")
```

```
[3]:      C      D
A
bar -1.510719 -0.197303
foo  0.986110  0.187550
```

1.0.126 分组计算每一列的分位数

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("quantile", q=0.75)
```

```
[3]:
```

	C	D
A		
bar	-0.842363	-0.316712
foo	1.624345	1.133769

1.0.127 分组计算每一列的均值标准误

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("sem")
```

```
[3]:
```

	C	D
A		
bar	0.504285	0.582442
foo	0.527038	0.468675

1.0.128 分组计算样本量

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("size")
```

```
[3]: A
bar    3
foo    5
dtype: int64
```


1.0.129 分组计算每一列的标准差

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("std")
```

```
[3]:
```

	C	D
A		
bar	0.873448	1.008819
foo	1.178492	1.047988

1.0.130 分组计算每一列的方差

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate("var")
```

```
[3]:
```

	C	D
A		
bar	0.762911	1.017716
foo	1.388843	1.098279

1.0.131 分组计算每一列的极差（自定义函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 定义计算极差的函数
def f(x):
    """
    x 是 Series 对象
    """
    res = x.max()-x.min()
    return res
```

```
[4]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped[["C", "D"]].aggregate(f)
```

```
[4]:
```

	C	D
A		

```
bar  1.689782  1.810770
foo  2.506019  2.561999
```

1.0.132 分组计算每一列的多个统计量（同时计算）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 定义计算极差的函数
def f(x):
    """
    x 是 Series 对象
    """
    res = x.max()-x.min()
    return res
```

```
[4]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
```

```
grouped[["C", "D"]].aggregate([f, "mean", "max", "var"])
```

```
[4]:
```

	C				D			
	f	mean	max	var	f	mean	max	
A								
bar	1.689782	-1.328755	-0.611756	0.762911	1.810770	-0.897855	-0.249370	
foo	2.506019	0.589037	1.744812	1.388843	2.561999	0.298522	1.462108	


```
var
```

	var
A	
bar	1.017716
foo	1.098279

1.0.133 分组计算每一列的多个统计量（同时计算，修改 Agg 的函数名）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
df
```

```
[2]:
```

	A	B	C	D
0	foo	one	1.624345	0.319039
1	bar	one	-0.611756	-0.249370
2	foo	two	-0.528172	1.462108
3	bar	three	-1.072969	-2.060141
4	foo	two	0.865408	-0.322417
5	bar	two	-2.301539	-0.384054
6	foo	one	1.744812	1.133769
7	foo	three	-0.761207	-1.099891

```
[3]: # 定义计算极差的函数
def f(x):
    """
    x 是 Series 对象
    """
    res = x.max()-x.min()
    return res
```

```
[4]: # 以 A 分组
grouped = df.groupby("A")
# 对 C D 求最大值的下标
grouped.agg(
    C 列最小值=pd.NamedAgg(column="C", aggfunc="min"),
    C 列极差=pd.NamedAgg(column="C", aggfunc=f),
    C 列平均值=pd.NamedAgg(column="C", aggfunc="mean"),
    D 列最小值=pd.NamedAgg(column="D", aggfunc="min"),
    D 列极差=pd.NamedAgg(column="D", aggfunc=f),
    D 列平均值=pd.NamedAgg(column="D", aggfunc="mean"),
)
```

```
[4]:          C 列最小值      C 列极差      C 列平均值      D 列最小值      D 列极差      D 列
平均值
A
bar -2.301539   1.689782 -1.328755 -2.060141   1.810770 -0.897855
foo -0.761207   2.506019   0.589037 -1.099891   2.561999   0.298522
```

1.0.134 分组计算每一列的多个统计量（同时计算，不同的列应用不同的函数）

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
        "D": np.random.randn(8),
    }
)
```

```
df
```

```
[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891
```

```
[3]: # 定义计算极差的函数
def f(x):
    """
    x 是 Series 对象
    """
    res = x.max()-x.min()
    return res
```

```
[4]: # 以 A 分组
grouped = df.groupby("A")
grouped.agg({"C": "sum", "D": "std"})
```

```
[4]:      C      D
A
bar -3.986264  1.008819
foo  2.945186  1.047988
```

1.0.135 分组计算每一列的累积计数

```
[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl
```

```
[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
```

```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:
   A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("cumsum")

```

```

[3]:
   C      D
0  1.624345  0.319039
1 -0.611756 -0.249370
2  1.096174  1.781147
3 -1.684725 -2.309511
4  1.961581  1.458730
5 -3.986264 -2.693565
6  3.706393  2.592499
7  2.945186  1.492608

```

1.0.136 分组计算每一列的累积最大值

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
    }
)

```



```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("cummax")

```

```

[3]:      C      D
0  1.624345  0.319039
1 -0.611756 -0.249370
2  1.624345  1.462108
3 -0.611756 -0.249370
4  1.624345  1.462108
5 -0.611756 -0.249370
6  1.744812  1.462108
7  1.744812  1.462108

```

1.0.137 分组计算每一列的累积最小值

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
    }
)

```

```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:
   A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("cummin")

```

```

[3]:
   C      D
0  1.624345  0.319039
1 -0.611756 -0.249370
2 -0.528172  0.319039
3 -1.072969 -2.060141
4 -0.528172 -0.322417
5 -2.301539 -2.060141
6 -0.528172 -0.322417
7 -0.761207 -1.099891

```

1.0.138 分组计算每一列的累乘

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
    }
)

```

```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:
   A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo  two -0.528172  1.462108
3  bar three -1.072969 -2.060141
4  foo  two  0.865408 -0.322417
5  bar  two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("cumprod")

```

```

[3]:
   C      D
0  1.624345  0.319039
1 -0.611756 -0.249370
2 -0.857933  0.466470
3  0.656395  0.513738
4 -0.742462 -0.150398
5 -1.510719 -0.197303
6 -1.295457 -0.170516
7  0.986110  0.187550

```

1.0.139 分组计算每一列的累积求和

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
    }
)

```

```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:
   A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo   two -0.528172  1.462108
3  bar  three -1.072969 -2.060141
4  foo   two  0.865408 -0.322417
5  bar   two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo  three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("cumsum")

```

```

[3]:
   C      D
0  1.624345  0.319039
1 -0.611756 -0.249370
2  1.096174  1.781147
3 -1.684725 -2.309511
4  1.961581  1.458730
5 -3.986264 -2.693565
6  3.706393  2.592499
7  2.945186  1.492608

```

1.0.140 分组计算每一列的差分

```

[1]: # 导入基础计算库
import numpy as np
# 导入数据分析库
import pandas as pd
# 导入 matplotlib 包
import matplotlib as mpl

```

```

[2]: # 设置随机数种子
np.random.seed(1)
# 生成数据
df = pd.DataFrame(
    {
        "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
        "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "C": np.random.randn(8),
    }
)

```

```

        "D": np.random.randn(8),
    }
)
df

```

```

[2]:      A      B      C      D
0  foo   one  1.624345  0.319039
1  bar   one -0.611756 -0.249370
2  foo  two -0.528172  1.462108
3  bar three -1.072969 -2.060141
4  foo  two  0.865408 -0.322417
5  bar  two -2.301539 -0.384054
6  foo   one  1.744812  1.133769
7  foo three -0.761207 -1.099891

```

```

[3]: # 以 A 分组
grouped = df.groupby("A")
grouped[["C", "D"]].transform("diff")

```

```

[3]:      C      D
0      NaN      NaN
1      NaN      NaN
2 -2.152517  1.143069
3 -0.461212 -1.810770
4  1.393579 -1.784525
5 -1.228570  1.676086
6  0.879404  1.456187
7 -2.506019 -2.233661

```

1.0.141 移动求和

```

[1]: import numpy as np
import pandas as pd

```

```

[2]: times = ['2020-01-01', '2020-01-03', '2020-01-04', '2020-01-05', '2020-01-29']
# 生成一个时间序列
s = pd.Series(range(5), index=pd.DatetimeIndex(times))
s

```

```

[2]: 2020-01-01    0
      2020-01-03    1
      2020-01-04    2
      2020-01-05    3
      2020-01-29    4
      dtype: int64

```

```
[3]: # 移动两期, 求和
      s.rolling(window=2).sum()
```

```
[3]: 2020-01-01    NaN
      2020-01-03    1.0
      2020-01-04    3.0
      2020-01-05    5.0
      2020-01-29    7.0
      dtype: float64
```

```
[4]: # 移动两天, 求和
      s.rolling(window='2D').sum()
```

```
[4]: 2020-01-01    0.0
      2020-01-03    1.0
      2020-01-04    3.0
      2020-01-05    5.0
      2020-01-29    4.0
      dtype: float64
```

1.0.142 移动平均, 中心化

```
[1]: import numpy as np
      import pandas as pd
```

```
[2]: times = ['2020-01-01', '2020-01-03', '2020-01-04', '2020-01-05', '2020-01-29']
      # 生成一个时间序列
      s = pd.Series(range(5), index=pd.DatetimeIndex(times))
      s
```

```
[2]: 2020-01-01    0
      2020-01-03    1
      2020-01-04    2
      2020-01-05    3
      2020-01-29    4
      dtype: int64
```

```
[3]: s.rolling(window=5).mean()
```

```
[3]: 2020-01-01    NaN
      2020-01-03    NaN
      2020-01-04    NaN
      2020-01-05    NaN
      2020-01-29    2.0
      dtype: float64
```

```
[4]: s.rolling(window=5, center=True).mean()
```

```
[4]: 2020-01-01    NaN
      2020-01-03    NaN
      2020-01-04    2.0
      2020-01-05    NaN
      2020-01-29    NaN
      dtype: float64
```

1.0.143 生成时间戳和事件跨度

```
[1]: import numpy as np
      import pandas as pd

[2]: pd.Timestamp("2012-05-01")

[2]: Timestamp('2012-05-01 00:00:00')

[3]: pd.Timestamp(2012, 5, 1)

[3]: Timestamp('2012-05-01 00:00:00')

[4]: pd.Period("2011-01")

[4]: Period('2011-01', 'M')

[5]: pd.Period("2012-05", freq="D")

[5]: Period('2012-05-01', 'D')
```

1.0.144 生成一个时间序列

```
[1]: import numpy as np
      import pandas as pd

[2]: dates = [
      pd.Timestamp("2012-05-01"),
      pd.Timestamp("2012-05-02"),
      pd.Timestamp("2012-05-03"),
      ]
      # 创建时间序列
      ts = pd.Series(np.random.randn(3), dates)

[3]: # 这个时间序列的索引是 DatetimeIndex 类型
      type(ts.index)

[3]: pandas.core.indexes.datetimes.DatetimeIndex

[4]: ts.index
```

```
[4]: DatetimeIndex(['2012-05-01', '2012-05-02', '2012-05-03'],
dtype='datetime64[ns]', freq=None)
```

```
[5]: ts
```

```
[5]: 2012-05-01    0.573656
      2012-05-02    1.003672
      2012-05-03    0.026830
      dtype: float64
```

```
[6]: # 创建一个时间跨度
      periods = [pd.Period("2012-01"), pd.Period("2012-02"), pd.Period("2012-03")]
      # 创建时间序列
      ts = pd.Series(np.random.randn(3), periods)
```

```
[7]: # 改时间序列的索引是 PeriodIndex 类型
      type(ts.index)
```

```
[7]: pandas.core.indexes.period.PeriodIndex
```

```
[8]: ts.index
```

```
[8]: PeriodIndex(['2012-01', '2012-02', '2012-03'], dtype='period[M]')
```

```
[9]: ts
```

```
[9]: 2012-01    0.124377
      2012-02    0.244281
      2012-03    1.218229
      Freq: M, dtype: float64
```

1.0.145 将字符串日期转为时间戳

```
[1]: import numpy as np
      import pandas as pd
```

```
[2]: pd.to_datetime(pd.Series(["Jul 31, 2009", "Jan 10, 2010", None]))
```

```
[2]: 0    2009-07-31
      1    2010-01-10
      2           NaT
      dtype: datetime64[ns]
```

```
[3]: pd.to_datetime(["2005/11/23", "2010/12/31"])
```

```
[3]: DatetimeIndex(['2005-11-23', '2010-12-31'], dtype='datetime64[ns]', freq=None)
```

```
[4]: pd.to_datetime("2010/11/12")
```

```
[4]: Timestamp('2010-11-12 00:00:00')
```



```
[5]: pd.Timestamp("2010/11/12")
```

```
[5]: Timestamp('2010-11-12 00:00:00')
```

```
[6]: pd.DatetimeIndex(["2018-01-01", "2018-01-03", "2018-01-05"])
```

```
[6]: DatetimeIndex(['2018-01-01', '2018-01-03', '2018-01-05'],
dtype='datetime64[ns]', freq=None)
```

1.0.146 将字符串日期转为时间戳，指定格式

```
[1]: import numpy as np
import pandas as pd
```

```
[2]: pd.to_datetime("2010/11/12", format="%Y/%m/%d")
```

```
[2]: Timestamp('2010-11-12 00:00:00')
```

```
[3]: pd.to_datetime("12-11-2010 00:00", format="%d-%m-%Y %H:%M")
```

```
[3]: Timestamp('2010-11-12 00:00:00')
```

```
[4]: df = pd.DataFrame(
    {"year": [2015, 2016], "month": [2, 3], "day": [4, 5], "hour": [2, 3]}
)
df
```

```
[4]:   year  month  day  hour
0  2015     2    4     2
1  2016     3    5     3
```

```
[5]: pd.to_datetime(df)
```

```
[5]: 0    2015-02-04 02:00:00
1    2016-03-05 03:00:00
dtype: datetime64[ns]
```

1.0.147 将秒数(毫秒, 纳秒, 微秒)转为时间戳

这些秒是从 origin 参数指定的时间计算的，默认是 1970 年 1 月 1 日零时

```
[1]: import numpy as np
import pandas as pd
```

```
[2]: pd.to_datetime(
    [1349720105, 1349806505, 1349892905, 1349979305, 1350065705], unit="s" # 秒
)
```

```
[2]: DatetimeIndex(['2012-10-08 18:15:05', '2012-10-09 18:15:05',
                    '2012-10-10 18:15:05', '2012-10-11 18:15:05',
```

```
'2012-10-12 18:15:05'],
dtype='datetime64[ns]', freq=None)
```

```
[3]: pd.to_datetime(
      [1349720105100, 1349720105200, 1349720105300, 1349720105400, 1349720105500],
      unit="ms", # 毫秒
    )
```

```
[3]: DatetimeIndex(['2012-10-08 18:15:05.100000', '2012-10-08 18:15:05.200000',
                    '2012-10-08 18:15:05.300000', '2012-10-08 18:15:05.400000',
                    '2012-10-08 18:15:05.500000'],
                    dtype='datetime64[ns]', freq=None)
```

```
[4]: pd.Timestamp(12623472000000000000)
```

```
[4]: Timestamp('2010-01-01 12:00:00')
```

```
[5]: pd.DatetimeIndex([12623472000000000000])
```

```
[5]: DatetimeIndex(['2010-01-01 12:00:00'], dtype='datetime64[ns]', freq=None)
```

```
[6]: pd.to_datetime([1490195805.433, 1490195805.433502912], unit="s")
```

```
[6]: DatetimeIndex(['2017-03-22 15:16:45.433000088', '2017-03-22
                    15:16:45.433502913'], dtype='datetime64[ns]', freq=None)
```

```
[7]: pd.to_datetime(1490195805433502912, unit="ns")
```

```
[7]: Timestamp('2017-03-22 15:16:45.433502912')
```

1.0.148 将时间戳转为秒数(毫秒, 纳秒, 微秒)

这些秒是从 origin 参数指定的时间计算的, 默认是 1970 年 1 月 1 日零时

```
[1]: import numpy as np
      import pandas as pd
```

```
[2]: stamps = pd.date_range("2012-10-08 18:15:05", periods=4, freq="D")
      stamps
```

```
[2]: DatetimeIndex(['2012-10-08 18:15:05', '2012-10-09 18:15:05',
                    '2012-10-10 18:15:05', '2012-10-11 18:15:05'],
                    dtype='datetime64[ns]', freq='D')
```

```
[3]: (stamps - pd.Timestamp("1970-01-01")) // pd.Timedelta("1s")
```

```
[3]: Int64Index([1349720105, 1349806505, 1349892905, 1349979305], dtype='int64')
```

1.0.149 生成一个时间戳序列 (date_range)

```
[1]: import numpy as np
import pandas as pd
```

```
[2]: import datetime
start = datetime.datetime(2011, 1, 1)
end = datetime.datetime(2012, 1, 1)
index = pd.date_range(start, end)
index
```

```
[2]: DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-04',
                    '2011-01-05', '2011-01-06', '2011-01-07', '2011-01-08',
                    '2011-01-09', '2011-01-10',
                    ...,
                    '2011-12-23', '2011-12-24', '2011-12-25', '2011-12-26',
                    '2011-12-27', '2011-12-28', '2011-12-29', '2011-12-30',
                    '2011-12-31', '2012-01-01'],
                    dtype='datetime64[ns]', length=366, freq='D')
```

```
[3]: # 工作日时间
index = pd.bdate_range(start, end)
index
```

```
[3]: DatetimeIndex(['2011-01-03', '2011-01-04', '2011-01-05', '2011-01-06',
                    '2011-01-07', '2011-01-10', '2011-01-11', '2011-01-12',
                    '2011-01-13', '2011-01-14',
                    ...,
                    '2011-12-19', '2011-12-20', '2011-12-21', '2011-12-22',
                    '2011-12-23', '2011-12-26', '2011-12-27', '2011-12-28',
                    '2011-12-29', '2011-12-30'],
                    dtype='datetime64[ns]', length=260, freq='B')
```

```
[4]: # 月份为间隔
pd.date_range(start, periods=1000, freq="M")
```

```
[4]: DatetimeIndex(['2011-01-31', '2011-02-28', '2011-03-31', '2011-04-30',
                    '2011-05-31', '2011-06-30', '2011-07-31', '2011-08-31',
                    '2011-09-30', '2011-10-31',
                    ...,
                    '2093-07-31', '2093-08-31', '2093-09-30', '2093-10-31',
                    '2093-11-30', '2093-12-31', '2094-01-31', '2094-02-28',
                    '2094-03-31', '2094-04-30'],
                    dtype='datetime64[ns]', length=1000, freq='M')
```

```
[5]: # 季节为间隔, 工作日
pd.bdate_range(start, periods=250, freq="BQS")
```

```
[5]: DatetimeIndex(['2011-01-03', '2011-04-01', '2011-07-01', '2011-10-03',
                    '2012-01-02', '2012-04-02', '2012-07-02', '2012-10-01',
                    '2013-01-01', '2013-04-01',
                    ...
                    '2071-01-01', '2071-04-01', '2071-07-01', '2071-10-01',
                    '2072-01-01', '2072-04-01', '2072-07-01', '2072-10-03',
                    '2073-01-02', '2073-04-03'],
                    dtype='datetime64[ns]', length=250, freq='BQS-JAN')
```

```
[6]: # 月份间隔
pd.date_range(start, end, freq="BM")
```

```
[6]: DatetimeIndex(['2011-01-31', '2011-02-28', '2011-03-31', '2011-04-29',
                    '2011-05-31', '2011-06-30', '2011-07-29', '2011-08-31',
                    '2011-09-30', '2011-10-31', '2011-11-30', '2011-12-30'],
                    dtype='datetime64[ns]', freq='BM')
```

```
[7]: # 星期为间隔
pd.date_range(start, end, freq="W")
```

```
[7]: DatetimeIndex(['2011-01-02', '2011-01-09', '2011-01-16', '2011-01-23',
                    '2011-01-30', '2011-02-06', '2011-02-13', '2011-02-20',
                    '2011-02-27', '2011-03-06', '2011-03-13', '2011-03-20',
                    '2011-03-27', '2011-04-03', '2011-04-10', '2011-04-17',
                    '2011-04-24', '2011-05-01', '2011-05-08', '2011-05-15',
                    '2011-05-22', '2011-05-29', '2011-06-05', '2011-06-12',
                    '2011-06-19', '2011-06-26', '2011-07-03', '2011-07-10',
                    '2011-07-17', '2011-07-24', '2011-07-31', '2011-08-07',
                    '2011-08-14', '2011-08-21', '2011-08-28', '2011-09-04',
                    '2011-09-11', '2011-09-18', '2011-09-25', '2011-10-02',
                    '2011-10-09', '2011-10-16', '2011-10-23', '2011-10-30',
                    '2011-11-06', '2011-11-13', '2011-11-20', '2011-11-27',
                    '2011-12-04', '2011-12-11', '2011-12-18', '2011-12-25',
                    '2012-01-01'],
                    dtype='datetime64[ns]', freq='W-SUN')
```

```
[8]: # 指定序列的长度，从结尾日期开始
pd.bdate_range(end=end, periods=20)
```

```
[8]: DatetimeIndex(['2011-12-05', '2011-12-06', '2011-12-07', '2011-12-08',
                    '2011-12-09', '2011-12-12', '2011-12-13', '2011-12-14',
                    '2011-12-15', '2011-12-16', '2011-12-19', '2011-12-20',
                    '2011-12-21', '2011-12-22', '2011-12-23', '2011-12-26',
                    '2011-12-27', '2011-12-28', '2011-12-29', '2011-12-30'],
                    dtype='datetime64[ns]', freq='B')
```

```
[9]: # 指定序列的长度, 从开始日期开始
pd.bdate_range(start=start, periods=20)
```

```
[9]: DatetimeIndex(['2011-01-03', '2011-01-04', '2011-01-05', '2011-01-06',
                    '2011-01-07', '2011-01-10', '2011-01-11', '2011-01-12',
                    '2011-01-13', '2011-01-14', '2011-01-17', '2011-01-18',
                    '2011-01-19', '2011-01-20', '2011-01-21', '2011-01-24',
                    '2011-01-25', '2011-01-26', '2011-01-27', '2011-01-28'],
                    dtype='datetime64[ns]', freq='B')
```

```
[10]: # 字符串日期指定起始
pd.date_range("2018-01-01", "2018-01-05", periods=5)
```

```
[10]: DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
                    '2018-01-05'],
                    dtype='datetime64[ns]', freq=None)
```

```
[11]: pd.date_range("2018-01-01", "2018-01-05", periods=10)
```

```
[11]: DatetimeIndex(['2018-01-01 00:00:00', '2018-01-01 10:40:00',
                    '2018-01-01 21:20:00', '2018-01-02 08:00:00',
                    '2018-01-02 18:40:00', '2018-01-03 05:20:00',
                    '2018-01-03 16:00:00', '2018-01-04 02:40:00',
                    '2018-01-04 13:20:00', '2018-01-05 00:00:00'],
                    dtype='datetime64[ns]', freq=None)
```

1.0.150 按照时间索引提取元素

```
[1]: import numpy as np
import pandas as pd
```

```
[2]: start="2010-01-01"
end="2011-12-31"
rng = pd.date_range(start, end, freq="BM")
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts.index
```

```
[2]: DatetimeIndex(['2010-01-29', '2010-02-26', '2010-03-31', '2010-04-30',
                    '2010-05-31', '2010-06-30', '2010-07-30', '2010-08-31',
                    '2010-09-30', '2010-10-29', '2010-11-30', '2010-12-31',
                    '2011-01-31', '2011-02-28', '2011-03-31', '2011-04-29',
                    '2011-05-31', '2011-06-30', '2011-07-29', '2011-08-31',
                    '2011-09-30', '2011-10-31', '2011-11-30', '2011-12-30'],
                    dtype='datetime64[ns]', freq='BM')
```

```
[3]: # 以字符串索引
ts["1/31/2011"]
```

[3]: -0.5482838196964798

```
[4]: import datetime
      ts[datetime.datetime(2011, 12, 25):]
```

[4]: 2011-12-30 0.615083
Freq: BM, dtype: float64

```
[5]: # 日期字符串切片
      ts["10/31/2011":"12/31/2011"]
```

[5]: 2011-10-31 0.094092
 2011-11-30 -1.676836
 2011-12-30 0.615083
Freq: BM, dtype: float64

```
[6]: # 提取月份, 2011 年的所有月份
      ts["2011"]
```

[6]: 2011-01-31 -0.548284
 2011-02-28 -1.238285
 2011-03-31 -1.445370
 2011-04-29 -1.405894
 2011-05-31 -1.149388
 2011-06-30 0.077598
 2011-07-29 -1.874590
 2011-08-31 -0.026948
 2011-09-30 -0.110488
 2011-10-31 0.094092
 2011-11-30 -1.676836
 2011-12-30 0.615083
Freq: BM, dtype: float64

```
[7]: # 提取 6 月份的最后一天
      ts["2011-6"]
```

[7]: 2011-06-30 0.077598
Freq: BM, dtype: float64

1.0.151 获取时间的年月日时分秒

```
[1]: import numpy as np
      import pandas as pd
```

```
[2]: idx = pd.date_range(start="2019-12-29 12:13:14", freq="s", periods=4000000)
      # 转为 Series 对象
      idx = pd.Series(idx)
      idx
```

```
[2]: 0      2019-12-29 12:13:14
      1      2019-12-29 12:13:15
      2      2019-12-29 12:13:16
      3      2019-12-29 12:13:17
      4      2019-12-29 12:13:18
      ...
      3999995 2020-02-13 19:19:49
      3999996 2020-02-13 19:19:50
      3999997 2020-02-13 19:19:51
      3999998 2020-02-13 19:19:52
      3999999 2020-02-13 19:19:53
      Length: 4000000, dtype: datetime64[ns]
```

```
[3]: # 年份
      idx.dt.year
```

```
[3]: 0      2019
      1      2019
      2      2019
      3      2019
      4      2019
      ...
      3999995 2020
      3999996 2020
      3999997 2020
      3999998 2020
      3999999 2020
      Length: 4000000, dtype: int64
```

```
[4]: # 月份
      idx.dt.month
```

```
[4]: 0      12
      1      12
      2      12
      3      12
      4      12
      ..
      3999995 2
      3999996 2
      3999997 2
      3999998 2
      3999999 2
      Length: 4000000, dtype: int64
```

```
[5]: # 日子
      idx.dt.day
```

```
[5]: 0      29
      1      29
      2      29
      3      29
      4      29
      ..
      3999995  13
      3999996  13
      3999997  13
      3999998  13
      3999999  13
      Length: 4000000, dtype: int64
```

```
[6]: # 小时
      idx.dt.hour
```

```
[6]: 0      12
      1      12
      2      12
      3      12
      4      12
      ..
      3999995  19
      3999996  19
      3999997  19
      3999998  19
      3999999  19
      Length: 4000000, dtype: int64
```

```
[7]: # 分钟
      idx.dt.minute
```

```
[7]: 0      13
      1      13
      2      13
      3      13
      4      13
      ..
      3999995  19
      3999996  19
      3999997  19
      3999998  19
      3999999  19
      Length: 4000000, dtype: int64
```

```
[8]: # 秒
      idx.dt.second
```



```
[8]: 0      14
      1      15
      2      16
      3      17
      4      18
      ..
      3999995  49
      3999996  50
      3999997  51
      3999998  52
      3999999  53
      Length: 4000000, dtype: int64
```

```
[9]: # 日期
      idx.dt.date
```

```
[9]: 0      2019-12-29
      1      2019-12-29
      2      2019-12-29
      3      2019-12-29
      4      2019-12-29
      ...
      3999995  2020-02-13
      3999996  2020-02-13
      3999997  2020-02-13
      3999998  2020-02-13
      3999999  2020-02-13
      Length: 4000000, dtype: object
```

```
[10]: # 时间
      idx.dt.time
```

```
[10]: 0      12:13:14
      1      12:13:15
      2      12:13:16
      3      12:13:17
      4      12:13:18
      ...
      3999995  19:19:49
      3999996  19:19:50
      3999997  19:19:51
      3999998  19:19:52
      3999999  19:19:53
      Length: 4000000, dtype: object
```