Pandas实例

构造杂乱数据

插值处理

字符串分割

标准化字符串

数据合并

文本匹配

列表展开

1.基本操作

pandas库的版本

In [1]:

```
import numpy as np
import pandas as pd
import warnings

# 打印出pandas 库的版本信息
pd.__version__
```

Out[1]:

```
'1.4.1'
```

构造DataFrame

In [2]:

```
# 数据字典data 以及列表格式的标签数据labels
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog',
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

In [3]:

```
df = pd.DataFrame(data, index=labels)
df
```

Out[3]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no

	animal	age	visits	priority
j	dog	3.0	1	no

数据基本信息

In [4]:

```
df.info()  # 数据相关的基本信息
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   animal    10 non-null     object
 1   age       8 non-null      float64
 2   visits    10 non-null     int64
 3   priority  10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

Out[4]:

	age	visits
count	8.000000	10.000000
mean	3.437500	1.900000
std	2.007797	0.875595
min	0.500000	1.000000
25%	2.375000	1.000000
50%	3.000000	2.000000
75%	4.625000	2.750000
max	7.000000	3.000000

选择列

In [5]:

```
# 从 DataFrame `df` 选择标签为 `animal` 和 `age` 的列
df[["animal","age"]]
```

Out[5]:

	animal	age
a	cat	2.5
b	cat	3.0
c	snake	0.5
d	dog	NaN
e	dog	5.0
f	cat	2.0
g	snake	4.5
h	cat	NaN
i	dog	7.0
j	dog	3.0

```
In [6]:  df.loc[:,["animal","age"]]
```

```
Out[6]:
     animal  age
a       cat  2.5
b       cat  3.0
c     snake  0.5
d       dog  NaN
e       dog  5.0
f       cat  2.0
g     snake  4.5
h       cat  NaN
i       dog  7.0
j       dog  3.0
```

选择行和列

```
In [7]:  # 在 [3, 4, 8] 行中，列为 ['animal', 'age'] 的数据
         df.loc[df.index[[3, 4, 8]], ['animal', 'age']]
```

```
Out[7]:
     animal  age
d       dog  NaN
e       dog  5.0
i       dog  7.0
```

```
In [8]:  df.iloc[[3, 4, 8], [0, 1]]
```

```
Out[8]:
     animal  age
d       dog  NaN
e       dog  5.0
i       dog  7.0
```

数据筛选

```
In [9]:  # 选择列```visits``` 大于 1 的行
         df[df['visits'] > 1]
```

```
Out[9]:
     animal  age  visits  priority
b       cat  3.0       3       yes
c     snake  0.5       2        no
d       dog  NaN       3       yes
e       dog  5.0       2        no
```

	animal	age	visits	priority
f	cat	2.0	3	no
i	dog	7.0	2	no

In [10]:

```
# 选择 `age` 为缺失值的行
df[df['age'].isnull()]
```

Out[10]:

	animal	age	visits	priority
d	dog	NaN	3	yes
h	cat	NaN	1	yes

In [11]:

```
# 选择 `animal` 是cat且`age` 小于 3 的行
df[(df['animal'] == 'cat') & (df['age'] < 3)]
```

Out[11]:

	animal	age	visits	priority
a	cat	2.5	1	yes
f	cat	2.0	3	no

In [12]:

```
# 选择 `age` 在 2 到 4 之间的数据（包含边界值）
df[df['age'].between(2, 4)]
```

Out[12]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
f	cat	2.0	3	no
j	dog	3.0	1	no

修改元素值

In [13]:

```
# 将 'f' 行的 `age` 改为 1.5
df.loc['f', 'age'] = 1.5
```

列求和

In [14]:

```
# 对 `visits` 列的数据求和
df['visits'].sum()
```

Out[14]:

```
19
```

分组求和

In [15]:

```
# 计算每种 `animal` `age` 的和
df.groupby('animal')['age'].sum()
```

Out[15]:

```
animal
cat       7.0
```

```
dog       15.0
snake      5.0
Name: age, dtype: float64
```

增加行

In [16]:

```
# 新增一行数据 k，数据自定义
df.loc['k'] = [5.5, 'dog', 'no', 2]
df
```

Out[16]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	1.5	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no
k	5.5	dog	no	2

删除行

In [17]:

```
# 删除新追加的 k 行
df = df.drop('k', axis=0)
df
```

Out[17]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	1.5	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

个数统计

In [18]:

```
# 统计每种 `animal` 的个数
```

```
df['animal'].value_counts()
```

Out[18]:

```
cat      4
dog      4
snake    2
Name: animal, dtype: int64
```

列排序

In [19]:

```
# 先根据 `age` 降序排列，再根据 `visits` 升序排列
df.sort_values(by=['age', 'visits'], ascending=[False, True])  # 默认升序
```

Out[19]:

	animal	age	visits	priority
i	dog	7.0	2	no
e	dog	5.0	2	no
g	snake	4.5	1	no
j	dog	3.0	1	no
b	cat	3.0	3	yes
a	cat	2.5	1	yes
f	cat	1.5	3	no
c	snake	0.5	2	no
h	cat	NaN	1	yes
d	dog	NaN	3	yes

数据转换map

In [20]:

```
# 将 `priority` 列的 `yes` 和 `no` 用 `True` 和 `False` 替换
df['priority'] = df['priority'].map({'yes': True, 'no': False})  # 使用map方法将字符
df
```

Out[20]:

	animal	age	visits	priority
a	cat	2.5	1	True
b	cat	3.0	3	True
c	snake	0.5	2	False
d	dog	NaN	3	True
e	dog	5.0	2	False
f	cat	1.5	3	False
g	snake	4.5	1	False
h	cat	NaN	1	True
i	dog	7.0	2	False
j	dog	3.0	1	False

数据替换replace

In [21]:

```
# 将 `animal` 列的 `snake` 用 `python` 替换
```

```
df['animal'] = df['animal'].replace('snake', 'python')
df
```

Out[21]:

	animal	age	visits	priority
a	cat	2.5	1	True
b	cat	3.0	3	True
c	python	0.5	2	False
d	dog	NaN	3	True
e	dog	5.0	2	False
f	cat	1.5	3	False
g	python	4.5	1	False
h	cat	NaN	1	True
i	dog	7.0	2	False
j	dog	3.0	1	False

In [22]:

```
# df['animal'] = df['animal'].map({'python': 'snake'})  # map未写完类别，其余的都被置
# df
```

数据透视表

In [23]:

```
# 对于每种 `animal` 和 `visit`，求出平均年龄。换句话说，每一行都是动物，每一列都是访问
df.pivot_table(index='animal', columns='visits', values='age', aggfunc='mean')
```

Out[23]:

visits	**1**	**2**	**3**
animal			
cat	2.5	NaN	2.25
dog	3.0	6.0	NaN
python	4.5	0.5	NaN

2.搭配操作

In [24]:

```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
df
```

Out[24]:

	A
0	1
1	2
2	2
3	3
4	4
5	5

	A
6	5
7	5
8	6
9	7
10	7

列元素去重

In [25]:

```
# `df`中`A`列出现的元素的唯一值（即：出现过的所有元素的集合）
df['A'].unique()
```

Out[25]:

```
array([1, 2, 3, 4, 5, 6, 7], dtype=int64)
```

数据去重

In [26]:

```
# 将`df`进行数据降重
df.drop_duplicates(['A'])  # 只保留第一次出现的行
```

Out[26]:

	A
0	1
1	2
3	3
4	4
5	5
8	6
9	7

列相减

In [27]:

```
# 给定一组随机数据
df = pd.DataFrame(np.random.random(size=(5, 3)))
df
```

Out[27]:

	0	1	2
0	0.374783	0.263837	0.992780
1	0.479938	0.981076	0.838327
2	0.099663	0.022200	0.671596
3	0.954756	0.485253	0.523484
4	0.857240	0.043054	0.138441

In [28]:

```
# 使每个元素减去所在行的平均值
df.sub(df.mean(axis=1), axis=0)
```

Out[28]:

	0	1	2
0	-0.169017	-0.279963	0.448980
1	-0.286509	0.214629	0.071880
2	-0.164823	-0.242287	0.407110
3	0.300258	-0.169245	-0.131013
4	0.510995	-0.303191	-0.207804

极值索引

In [29]:

```
df = pd.DataFrame(np.random.random(size=(5, 10)), columns=list(’abcdefghij’))
df
```

Out[29]:

	a	b	c	d	e	f	g	h	i	j
0	0.567944	0.921168	0.521974	0.583570	0.178107	0.903390	0.261774	0.836178	0.726512	0.586562
1	0.279541	0.242886	0.600041	0.520493	0.710574	0.654514	0.366569	0.540094	0.345685	0.516206
2	0.310480	0.386638	0.362324	0.292853	0.757898	0.217559	0.394976	0.232218	0.297622	0.499301
3	0.887586	0.049991	0.949864	0.731598	0.573696	0.145510	0.041429	0.615991	0.524794	0.811569
4	0.245327	0.642334	0.937530	0.272604	0.599459	0.966564	0.727655	0.300235	0.139796	0.749208

In [30]:

```
# 返回下列`df`数字总和最小那列的标签
# Pandas 里面的 idxmin 、idxmax函数与Numpy中 argmax、argmin 用法大致相同
df.sum().idxmin()  # idxmin()返回第一次出现的最小/最大值的索引
```

Out[30]:

```
’g’
```

In [31]:

```
df.columns[np.argmin(df.sum())]  # argmin返回第一次出现的最小/最大值的索引
```

Out[31]:

```
’g’
```

删除重复项

In [32]:

```
df = pd.DataFrame(np.random.randint(0, 2, size=(10, 3)))
df
```

Out[32]:

	0	1	2
0	0	1	1
1	1	1	0
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	1
6	1	0	1

	0	1	2
7	1	0	1
8	1	0	0
9	0	0	0

In [33]:

```
# 计算一个 DataFrame 有多少不重复的行
df.drop_duplicates(keep=False)  # keep=False 删除所有重复项
```

Out[33]:

	0	1	2
3	1	1	1
8	1	0	0
9	0	0	0

In [34]:

```
len(df.drop_duplicates(keep=False))
```

Out[34]:

```
3
```

数据分段分组

In [35]:

```
df = pd.DataFrame(np.random.RandomState(1).randint(1, 4, size=(10, 2)), columns = ["A
df
```

Out[35]:

	A	B
0	2	1
1	1	2
2	2	1
3	1	2
4	1	2
5	1	3
6	2	3
7	1	3
8	2	3
9	1	1

In [36]:

```
# 产生一个随机状态种子 np.random.RandomState(1).randint 左闭右开
df = pd.DataFrame(np.random.RandomState(1).randint(0, 101,
                  size=(100, 2)),
                  columns = ["A", "B"])
df
```

Out[36]:

	A	B
0	37	12

	A	B
1	72	9
2	75	5
3	79	64
4	16	1
...	...	...
95	71	53
96	69	36
97	21	40
98	77	91
99	49	47

100 rows × 2 columns

In [37]:

```
# 对 A 进行分段分组（i.e. (0, 10], (10, 20], ...），求每组内 B 的和。
df.groupby(pd.cut(df['A'], np.arange(0, 101, 10)))['B'].sum()  # pd.cut数据分段
```

Out[37]:

```
A
(0, 10]      752
(10, 20]     475
(20, 30]     684
(30, 40]     161
(40, 50]     384
(50, 60]     839
(60, 70]     428
(70, 80]     615
(80, 90]     358
(90, 100]    300
Name: B, dtype: int32
```

3.数据清洗

构造杂乱数据

In [38]:

```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
                               'Budapest_PaRis', 'Brussels_londOn'],
                   'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                   'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                        'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
                                    '12. Air France', '"Swiss Air"']})
df
```

Out[38]:

	From_To	FlightNumber	RecentDelays	Airline
0	LoNDon_paris	10045.0	[23, 47]	KLM(!)
1	MAdrid_miLAN	NaN	[]	<Air France> (12)
2	londON_StockhOlm	10065.0	[24, 43, 87]	(British Airways.)
3	Budapest_PaRis	NaN	[13]	12. Air France
4	Brussels_londOn	10085.0	[67, 32]	"Swiss Air"

插值处理

FlightNumber列中的某些值缺失（它们是NaN）。

这些数字是有规律的，即每行增加 10，因此 `NaN` 需要放置 10055 和 10075。

修改 `df` 以填充这些缺失的数字并使该列成为整数列（而不是浮点列）

df.interpolate()

DataFrame.interpolate(method='linear', axis=0, limit=None, inplace=False, limit_direction='forward', limit_area=None, downcast=None, **kwargs)

method插值方式：

nearest：最邻近插值法

zero：阶梯插值

slinear、linear：线性插值

quadratic、cubic：2、3阶B样条曲线插值

In [39]:

```
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)  # 插值函数
df
```

Out[39]:

	From_To	FlightNumber	RecentDelays	Airline
0	LoNDon_paris	10045	[23, 47]	KLM(!)
1	MAdrid_miLAN	10055	[]	<Air France> (12)
2	londON_StockhOlm	10065	[24, 43, 87]	(British Airways.)
3	Budapest_PaRis	10075	[13]	12. Air France
4	Brussels_londOn	10085	[67, 32]	"Swiss Air"

字符串分割

From_To列作为两个单独的列会更好！

拆分下划线分隔符 _ 前后的每个字符串. 将其拆分成两列，

存放在一个名为"temp"的临时 DataFrame，将列名 'From' 和 'To' 分配给这个临时DataFrame.

In [40]:

```
temp = df['From_To'].str.split('_', expand=True)
temp.columns = ['From', 'To']
temp
```

Out[40]:

	From	To
0	LoNDon	paris
1	MAdrid	miLAN
2	londON	StockhOlm
3	Budapest	PaRis

	From	To
4	Brussels	londOn

标准化字符串

注意城市名称的大小写是混合在一起的。

只有第一个字母是大写的（例如“londON”应该变成“London”。）

In [41]:

```
temp['From'] = temp['From'].str.capitalize()
temp['To'] = temp['To'].str.capitalize()
temp
```

Out[41]:

	From	To
0	London	Paris
1	Madrid	Milan
2	London	Stockholm
3	Budapest	Paris
4	Brussels	London

数据合并

将 `From_To` 列从 `df` 中删去，将 `temp` 处理好的数据合并到 `df` 中

In [42]:

```
df = df.drop('From_To', axis=1)
df = df.join(temp)
df
```

Out[42]:

	FlightNumber	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM(!)	London	Paris
1	10055	[]	<Air France> (12)	Madrid	Milan
2	10065	[24, 43, 87]	(British Airways.)	London	Stockholm
3	10075	[13]	12. Air France	Budapest	Paris
4	10085	[67, 32]	"Swiss Air"	Brussels	London

文本匹配

只提取航空公司名称。

在 `AirLine` 列中，您可以看到航空公司名称周围出现了一些额外的符号。

例如'(British Airways.)'应该变成'British Airways'.

In [43]:

```
df['Airline'] = df['Airline'].str.extract('([a-zA-Z\s]+)', expand=False).str.strip()
df
```

Out[43]:

	FlightNumber	RecentDelays	Airline	From	To

	FlightNumber	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM	London	Paris
1	10055	[]	Air France	Madrid	Milan
2	10065	[24, 43, 87]	British Airways	London	Stockholm
3	10075	[13]	Air France	Budapest	Paris
4	10085	[67, 32]	Swiss Air	Brussels	London

列表展开

在 RecentDelays 列中，值已作为列表输入到 DataFrame 中。我们希望每个第一个值在它自己的列中，每个第二个值在它自己的列中，依此类推。如果没有第 N 个值，则该值应为 NaN。

将 Series 列表展开为名为 的 DataFrame delays，重命名列delay_1，delay_2等等，并将不需要的RecentDelays 列替换df为delays

In [44]:

```
delays = df['RecentDelays'].apply(pd.Series)
delays
```

```
C:\Users\29511_orbf8\AppData\Local\Temp\ipykernel_54344\139781484.py:1: FutureWarning:
The default dtype for empty Series will be 'object' instead of 'float64' in a future v
ersion. Specify a dtype explicitly to silence this warning.
  delays = df['RecentDelays'].apply(pd.Series)
```

Out[44]:

	0	1	2
0	23.0	47.0	NaN
1	NaN	NaN	NaN
2	24.0	43.0	87.0
3	13.0	NaN	NaN
4	67.0	32.0	NaN

In [45]:

```
delays.columns = ['delay_{}'.format(n) for n in range(1, len(delays.columns)+1)]  #
df = df.drop('RecentDelays', axis=1).join(delays)  # 将新的列加入到原始数据中
df
```

Out[45]:

	FlightNumber	Airline	From	To	delay_1	delay_2	delay_3
0	10045	KLM	London	Paris	23.0	47.0	NaN
1	10055	Air France	Madrid	Milan	NaN	NaN	NaN
2	10065	British Airways	London	Stockholm	24.0	43.0	87.0
3	10075	Air France	Budapest	Paris	13.0	NaN	NaN
4	10085	Swiss Air	Brussels	London	67.0	32.0	NaN