Pandas 数据清洗

Guangyao Zhao

2022-12-10

Contents

```python
import pandas as pd
import numpy as np

df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': ['A', None, 'C', 'D'], 'C': [1, None, 3, 4]})
df
```

	A	B	C
0	1	A	1.0
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

数据清洗

缺失值类型

在处理缺失值之前先明确一下缺失值有哪些:

- None
- np.nan
- pd.NaT: pandas 中时间的空值类型

df.isna(), df.isnull()

两者等价, 都是对单元素进行判断, True 为空值:

```
df.isna()
```

	A	B	C
0	False	False	False
1	False	True	True
2	False	False	False
3	False	False	False

```
df.isnull()
```

	A	B	C
0	False	False	False
1	False	True	True
2	False	False	False
3	False	False	False

单独对某列进行判断：

```
df['A'].isna()
```

	A
0	False
1	False
2	False
3	False

判断每一列的空值总数：

```
df.isna().sum(axis = 0)
```

	0
A	0
B	1
C	1

缺失值统计

同样也可以判断每一行的空值总数：

```
df.isna().sum(axis = 1)
```

	0
0	0
1	2
2	0
3	0

判断每一列是否含有空值，即只要有 1 个空值就算有：

```
df.isna().any(axis = 0)
```

	0
A	False
B	True
C	True

判断每一列是否含有空值，即全是空值才算有：

```
df.isna().all(axis = 0)
```

	0
A	False
B	False
C	False

缺失值筛选

筛选出含有缺失值的行：

```
df.loc[df.isna().any(axis = 1), :]
```

	A	B	C
1	2	None	NaN

筛选出含有缺失值的列：

```
df.loc[:, df.isna().any(axis = 0)]
```

	B	C
0	A	1.0
1	None	NaN
2	C	3.0
3	D	4.0

但更多的情况是要去除含有缺失值的行:

```
df.loc[~df.isna().any(axis = 1), :] # 对判断条件取反
```

	A	B	C
0	1	A	1.0
2	3	C	3.0
3	4	D	4.0

缺失值的操作

有时候并不会直接删除含有空值的行或列，而是进行某种补充。

`fillna()`

直接用 0 填充

```
df.fillna(value = 0)
```

	A	B	C
0	1	A	1.0
1	2	0	0.0
2	3	C	3.0
3	4	D	4.0

也可以使用某种字符串填充

```
df.fillna(value = 'missing values')
```

	A	B	C
0	1	A	1.0
1	2	missing values	missing values
2	3	C	3.0
3	4	D	4.0

针对不同列进行不同替换：

```
values = {'A': 'AAA', 'B': 'BBB', 'C': 'CCC'}
df.fillna(value = values)
```

	A	B	C
0	1	A	1.0
1	2	BBB	CCC
2	3	C	3.0
3	4	D	4.0

在某些数据集中，前后数据可能会有一定的相关，或者说变化很小，此时可以直接用前向填充或后向填充：

```
df.fillna(method = 'ffill') # forward + fill
```

	A	B	C
0	1	A	1.0
1	2	A	1.0
2	3	C	3.0
3	4	D	4.0

```
df.fillna(method = 'bfill') # backward + fill
```

	A	B	C
0	1	A	1.0
1	2	C	3.0
2	3	C	3.0
3	4	D	4.0

还可以用种统计数据填充，比如用平均值：

```
df.fillna(value = df.mean(axis = 0)) # 相当于先得到 df 的列平均，然后将其填充
```

	A	B	C
0	1	A	1.000000
1	2	None	2.666667
2	3	C	3.000000
3	4	D	4.000000

df.interpolate()

不同于直接使用具体数值填充，插值填充可以从整体数据的某种规律来填充，比如最简单的线性填充，更多的填充方法参考官方文档

```
df.interpolate(method = 'linear')
```

	A	B	C
0	1	A	1.0
1	2	None	2.0
2	3	C	3.0
3	4	D	4.0

df.dropna()

在 @subsec-any 中介绍了怎么筛选出含有空值的行和列，用 `df.dropna()` 则可直接删除。

只要有一个空值，就删除行：

```
df.dropna(axis = 0, how = 'any')
```

	A	B	C
0	1	A	1.0
2	3	C	3.0
3	4	D	4.0

全部是空值才删除行：

```
df.dropna(axis = 0, how = 'all')
```

	A	B	C
0	1	A	1.0
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

甚至可以给一个阈值，超过某个阈值后才删除：

```
df.dropna(axis = 0, thresh = 1)
```

	A	B	C
0	1	A	1.0
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

在某个子列中操作：

```
df.dropna(axis = 0, subset = ['A', 'C'])
```

	A	B	C
0	1	A	1.0
2	3	C	3.0
3	4	D	4.0

数据替换

df.replace()

数值替换对文本类型特别实用，整型浮点型很少使用。

将整个数据框的某个数值替换为另一个数值：

```
df.replace({1: 'new value'})
```

	A	B	C
0	new value	A	new value
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

根据列，进行不同的替换。将列 `'A'`, `'B'` 的值 `1`, `'A'` 替换为 `new value`:

```
df.replace({'A': 1, 'B': 'A'}, 'new value')
```

	A	B	C
0	new value	new value	1.0
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

除了固定的具体值外，可以指定具体的填充方法:

```
df.replace([1, 'A'], method = 'bfill')
```

	A	B	C
0	2	None	NaN
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

> Tip
>
> 对于字符的替换可以学习下正则化，利用正则化将字符按照一定的规律筛选出来，再进行替换。由于我基本不用字符类型的数据，在此就不献丑了。

df.clip()

有时候会要求数据有一定的大小范围，即大于某个阈值时，将其值替换为阈值，反之亦然。

```
df['A'].clip(lower = 1, upper = 2)
```

	A
0	1
1	2
2	2
3	2

重复值及删除数据

```
df = pd.DataFrame(data = {'A': [1, 1, 3], 'B': [2, 2, 3]})
df
```

	A	B
0	1	2
1	1	2
2	3	3

`df.duplicated()`

```
df.duplicated(keep = 'first', subset = ['A', 'B']) # subset 指的是可以指定列
```

	0
0	False
1	True
2	False

keep 的参数有：

- `first`：除第一次出现的外，重复的为 True
- `last`：除最后一次出现的外，重复的为 True
- `False`：只要出现重复的均标记为 True

`df.drop_duplicates()`

```
df.drop_duplicates(subset = ['A', 'B'], keep = 'first', inplace = False, ignore_index = True)
```

	A	B
0	1	2
1	3	3

df.drop()

以上均只能删除重复的行，并不能删除重复的列。而 df.drop() 可以指定行或列。

删除某些列：

```
df.drop(columns = ['B'])
```

	A
0	1
1	1
2	3

删除某些行：

```
df.drop(index = [0])
```

	A	B
1	1	2
2	3	3

Numpy 格式的转换

df.values 会返回 numpy 类型的值，但是不推荐这样使用。推荐使用：

- df.to_numpy()
- np.array()

```
type(df.to_numpy().dtype)
```

numpy.dtype[int64]

```
type(np.array(df).dtype)
```

```
numpy.dtype[int64]
```

文本处理

数据类型

字符的操作

.str 访问器

```
df = pd.DataFrame({'aaA_ test': [1, 2, 3, 4], '_BbC ': ['A', None, 'C', 'D'], 'Caa': [1, None, 3, 4]})
df
```

	aaA_ test	_BbC	Caa
0	1	A	1.0
1	2	None	NaN
2	3	C	3.0
3	4	D	4.0

```
df['_BbC '].astype(str).str
```

```
<pandas.core.strings.accessor.StringMethods at 0x127ae0be0>
```

将列 B 转换为小写：

```
df['_BbC '].astype(str).str.lower()
```

	_BbC
0	a
1	none
2	c
3	d

也可以将表头转换：

```
df.columns.str.lower()
```

```
Index(['aaa_ test', '_bbc ', 'caa'], dtype='object')
```

对字符串进行连续操作：

```
df.columns.str.strip().str.lower().str.replace('_', '__') # 去除空格；小写；替换下划线
```

```
Index(['aaa__ test', '__bbc', 'caa'], dtype='object')
```

文本格式

大写：

```
df.columns.str.upper()
```

```
Index(['AAA_ TEST', '_BBC ', 'CAA'], dtype='object')
```

标题格式，即每个单词的首字母大写：

```
df.columns.str.title()
```

```
Index(['Aaa_ Test', '_Bbc ', 'Caa'], dtype='object')
```

首字母大写：

```
df.columns.str.capitalize()
```

```
Index(['Aaa_ test', '_bbc ', 'Caa'], dtype='object')
```

大小写互换：

```
df.columns.str.swapcase()
```

```
Index(['AAa_ TEST', '_bBc ', 'cAA'], dtype='object')
```

文本对齐

居中：

```
df.columns.str.center(10, fillchar='*')
```

```
Index(['aaA_ test*', '**_BbC ***', '***Caa****'], dtype='object')
```

左对齐：

```
df.columns.str.ljust(10, fillchar='*') # left justify
```

```
Index(['aaA_ test*', '_BbC *****', 'Caa*******'], dtype='object')
```

指定宽度、对齐方式和填充内容：

```
df.columns.str.pad(width = 10, side = 'left', fillchar='*')
```

```
Index(['*aaA_ test', '*****_BbC ', '*******Caa'], dtype='object')
```

计数

计算每个字符串的长度：

```
df.columns.str.len()
```

```
Int64Index([9, 5, 3], dtype='int64')
```

编码：

```
df.columns.str.encode('utf-8')
```

```
Index([b'aaA_ test', b'_BbC ', b'Caa'], dtype='object')
```

格式判定

是否为字母：

```
df.columns.str.isalpha
```

```
<bound method _map_and_wrap.<locals>.wrapper of <pandas.core.strings.accessor.StringMethods object at 0x12
```

是否为数字 0-9：

```
df.columns.str.isnumeric
```

```
<bound method _map_and_wrap.<locals>.wrapper of <pandas.core.strings.accessor.StringMethods object at 0x12
```

是否为字母和数字：

```
df.columns.str.isalnum
```

```
<bound method _map_and_wrap.<locals>.wrapper of <pandas.core.strings.accessor.StringMethods object at 0x12
```

是否为数字:

```
df.columns.str.isdigit
```

```
<bound method _map_and_wrap.<locals>.wrapper of <pandas.core.strings.accessor.StringMethods object at 0x12
```

是否为小数：

```
df.columns.str.isdecimal
```

```
<bound method _map_and_wrap.<locals>.wrapper of <pandas.core.strings.accessor.StringMethods object at 0x12
```

文本的高级处理

```
s = pd.Series(['天 _ 地 _ 人 _ 狗', '你 _ 我 _ 他', np.nan, '风 _ 水 _ 火'], dtype=str)
s
```

	0
0	天 _ 地 _ 人 _ 狗
1	你 _ 我 _ 他
2	NaN
3	风 _ 水 _ 火

文本分隔

```
s.str.split('_')
```

	0
0	[天, 地, 人, 狗]
1	[你, 我, 他]
2	NaN
3	[风, 水, 火]

取出第 2 行：

```
s.str.split('_')[1]
```

```
['你', '我', '他']
```

取出最后一列：

```
s.str.split('_').str.get(-1)
```

	0
0	狗
1	他
2	NaN
3	火

切片：

```
s.str.split('_')[: 2]
```

	0
0	[天, 地, 人, 狗]
1	[你, 我, 他]

字符分隔展开

在 @subsec-split 中，可以将字符串切割开来，形成列表。但有时候我们常常希望能形成不同的列：

```
s.str.split('_', expand = True)
```

	0	1	2	3
0	天	地	人	狗
1	你	我	他	None
2	NaN	NaN	NaN	NaN
3	风	水	火	None

从右侧开始分隔：

```
s.str.rsplit('_', expand = True, n = 1)
```

	0	1
0	天_地_人	狗
1	你_我	他
2	NaN	NaN
3	风_水	火

文本替换

```
s = pd.Series(['10', '-¥20', '¥3,000'], dtype="string")

s.str.replace('¥|,', '') # 将 ¥ 和逗号删除
```

	0
0	10
1	-20
2	3000

指定替换

```
s = pd.Series(['ax', 'bxy', 'cxyz'])

s.str.slice_replace(stop=2, repl = 'T') # repl 是 replace 的
```

	0
0	T
1	Ty
2	Tyz

stop 表示到哪里为止，比如 stop = 2 表示将第一和二个字符替换掉。