# 数据分析 5 数据清洗和准备

在数据分析和建模的过程中，相当多的时间要用在数据准备上：加载、清理、转换以及重塑。这些工作会占到分析师时间的80%或更多。

pandas和内置的Python标准库提供了一组高级的、灵活的、快速的工具，可以让你轻松地将数据规整为想要的格式。

## 处理缺失数据

检测缺失数据

```
In [10]: string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])

In [11]: string_data
Out[11]:
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object

In [12]: string_data.isnull()
Out[12]:
0    False
1    False
2     True
3    False
dtype: bool
```

在统计应用中，NA数据可能是不存在的数据或者虽然存在，但是没有观察到（例如，数据采集中发生了问题）。当进行数据清洗以进行分析时，最好直接对缺失数据进行分析，以判断数据采集的问题或缺失数据可能导致的偏差。

Python内置的None值在对象数组中也可以作为NA：

```
In [13]: string_data[0] = None
```

```
In [14]: string_data.isnull()
Out[14]:
0     True
1    False
2     True
3    False
dtype: bool
```

## 滤除缺失数据

```
In [15]: from numpy import nan as NA

In [16]: data = pd.Series([1, NA, 3.5, NA, 7])

In [17]: data.dropna()
Out[17]:
0    1.0
2    3.5
4    7.0
dtype: float64
```

和这个效果一样

```
In [18]: data[data.notnull()]
Out[18]:
0    1.0
2    3.5
4    7.0
dtype: float64
```

DataFrame对象，dropna默认丢弃任何含有缺失值的行

```
In [19]: data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
   ....:                      [NA, NA, NA], [NA, 6.5, 3.]])
```

```
In [20]: cleaned = data.dropna()

In [21]: data
Out[21]:
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0

In [22]: cleaned
Out[22]:
     0    1    2
0  1.0  6.5  3.0
```

传入how='all'将只丢弃全为NA的那些行

```
In [23]: data.dropna(how='all')
Out[23]:
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
3  NaN  6.5  3.0
```

```
In [24]: data[4] = NA

In [25]: data
Out[25]:
     0    1    2   4
0  1.0  6.5  3.0 NaN
1  1.0  NaN  NaN NaN
2  NaN  NaN  NaN NaN
3  NaN  6.5  3.0 NaN

In [26]: data.dropna(axis=1, how='all')
Out[26]:
```

```
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
```

```
In [27]: df = pd.DataFrame(np.random.randn(7, 3))

In [28]: df.iloc[:4, 1] = NA

In [29]: df.iloc[:2, 2] = NA

In [30]: df
Out[30]:
          0         1         2
0 -0.204708       NaN       NaN
1 -0.555730       NaN       NaN
2  0.092908       NaN  0.769023
3  1.246435       NaN -1.296221
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741

In [31]: df.dropna()
Out[31]:
          0         1         2
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741

# 删除小于n个非空值的行
In [32]: df.dropna(thresh=2)
Out[32]:
          0         1         2
2  0.092908       NaN  0.769023
3  1.246435       NaN -1.296221
4  0.274992  0.228913  1.352917
```

```
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741
```

## 填充缺失数据

fillna方法是最主要的函数。通过一个常数调用fillna就会将缺失值替换为那个常数值

```
In [33]: df.fillna(0)
Out[33]:
          0         1         2
0 -0.204708  0.000000  0.000000
1 -0.555730  0.000000  0.000000
2  0.092908  0.000000  0.769023
3  1.246435  0.000000 -1.296221
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741
```

通过一个字典调用fillna，就可以实现对不同的列填充不同的值

```
In [34]: df.fillna({1: 0.5, 2: 0})
Out[34]:
          0         1         2
0 -0.204708  0.500000  0.000000
1 -0.555730  0.500000  0.000000
2  0.092908  0.500000  0.769023
3  1.246435  0.500000 -1.296221
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741
```

fillna默认会返回新对象，但也可以对现有对象进行就地修改

```
In [35]: _ = df.fillna(0, inplace=True)

In [36]: df
```

```
Out[36]:
          0         1         2
0 -0.204708  0.000000  0.000000
1 -0.555730  0.000000  0.000000
2  0.092908  0.000000  0.769023
3  1.246435  0.000000 -1.296221
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741
```

对reindexing有效的那些插值方法也可用于fillna

```
In [37]: df = pd.DataFrame(np.random.randn(6, 3))

In [38]: df.iloc[2:, 1] = NA

In [39]: df.iloc[4:, 2] = NA

In [40]: df
Out[40]:
          0         1         2
0  0.476985  3.248944 -1.021228
1 -0.577087  0.124121  0.302614
2  0.523772       NaN  1.343810
3 -0.713544       NaN -2.370232
4 -1.860761       NaN       NaN
5 -1.265934       NaN       NaN

In [41]: df.fillna(method='ffill')
Out[41]:
          0         1         2
0  0.476985  3.248944 -1.021228
1 -0.577087  0.124121  0.302614
2  0.523772  0.124121  1.343810
3 -0.713544  0.124121 -2.370232
4 -1.860761  0.124121 -2.370232
5 -1.265934  0.124121 -2.370232
```

```
In [42]: df.fillna(method='ffill', limit=2)
Out[42]:
          0         1         2
0  0.476985  3.248944 -1.021228
1 -0.577087  0.124121  0.302614
2  0.523772  0.124121  1.343810
3 -0.713544  0.124121 -2.370232
4 -1.860761       NaN -2.370232
5 -1.265934       NaN -2.370232
```

传入Series的平均值或中位数

```
In [43]: data = pd.Series([1., NA, 3.5, NA, 7])

In [44]: data.fillna(data.mean())
Out[44]:
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

## 数据转换

### 移除重复数据

```
In [45]: data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
   ....:                      'k2': [1, 1, 2, 3, 3, 4, 4]})

In [46]: data
Out[46]:
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
```

```
4  one   3
5  two   4
6  two   4
```

DataFrame的duplicated方法返回一个布尔型Series，表示各行是否是重复行

```
In [47]: data.duplicated()
Out[47]:
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

drop_duplicates方法，它会返回一个DataFrame，重复的数组会标为False

```
In [48]: data.drop_duplicates()
Out[48]:
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
```

只希望根据k1列过滤重复项

```
In [49]: data['v1'] = range(7)

In [50]: data.drop_duplicates(['k1'])
Out[50]:
    k1  k2  v1
```

```
0  one   1   0
1  two   1   1
```

duplicated和drop_duplicates默认保留的是第一个出现的值组合。传入keep='last'则保留最后一个

```
In [51]: data.drop_duplicates(['k1', 'k2'], keep='last')
Out[51]:
    k1  k2  v1
0  one   1   0
1  two   1   1
2  one   2   2
3  two   3   3
4  one   3   4
6  two   4   6
```

## 利用函数或映射进行数据转换

根据数组、Series或DataFrame列中的值来实现转换工作

```
In [52]: data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
   ....:                               'Pastrami', 'corned beef', 'Bacon',
   ....:                               'pastrami', 'honey ham', 'nova lox'],
   ....:                      'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})

In [53]: data
Out[53]:
          food  ounces
0        bacon     4.0
1  pulled pork     3.0
2        bacon    12.0
3     Pastrami     6.0
4  corned beef     7.5
5        Bacon     8.0
6     pastrami     3.0
7    honey ham     5.0
8     nova lox     6.0
```

```
```

添加一列表示该肉类食物来源的动物类型。我们先编写一个不同肉类到动物的映射

```
meat_to_animal = {
  'bacon': 'pig',
  'pulled pork': 'pig',
  'pastrami': 'cow',
  'corned beef': 'cow',
  'honey ham': 'pig',
  'nova lox': 'salmon'
}
```

使用Series的str.lower方法，将各个值转换为小写

```
In [55]: lowercased = data['food'].str.lower()

In [56]: lowercased
Out[56]:
0          bacon
1    pulled pork
2          bacon
3       pastrami
4    corned beef
5          bacon
6       pastrami
7      honey ham
8       nova lox
Name: food, dtype: object

In [57]: data['animal'] = lowercased.map(meat_to_animal)

In [58]: data
Out[58]:
          food  ounces  animal
0        bacon     4.0     pig
1  pulled pork     3.0     pig
2        bacon    12.0     pig
```

```
3     Pastrami     6.0     cow
4  corned beef     7.5     cow
5        Bacon     8.0     pig
6     pastrami     3.0     cow
7    honey ham     5.0     pig
8     nova lox     6.0  salmon
```

也可以传入一个能够完成全部这些工作的函数

```
In [59]: data['food'].map(lambda x: meat_to_animal[x.lower()])
Out[59]:
0       pig
1       pig
2       pig
3       cow
4       cow
5       pig
6       cow
7       pig
8    salmon
Name: food, dtype: object
```

## 替换值

```
In [60]: data = pd.Series([1., -999., 2., -999., -1000., 3.])

In [61]: data
Out[61]:
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
```

-999这个值可能是一个表示缺失数据的标记值。要将其替换为pandas能够理解的NA值

```
In [62]: data.replace(-999, np.nan)
Out[62]:
0        1.0
1        NaN
2        2.0
3        NaN
4    -1000.0
5        3.0
dtype: float64
```

一次性替换多个值

```
In [63]: data.replace([-999, -1000], np.nan)
Out[63]:
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

让每个值有不同的替换值，可以传递一个替换列表

```
In [64]: data.replace([-999, -1000], [np.nan, 0])
Out[64]:
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

```
In [65]: data.replace({-999: np.nan, -1000: 0})
Out[65]:
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

## 重命名轴索引

```
In [66]: data = pd.DataFrame(np.arange(12).reshape((3, 4)),
   ....:                     index=['Ohio', 'Colorado', 'New York'],
   ....:                     columns=['one', 'two', 'three', 'four'])
```

```
In [67]: transform = lambda x: x[:4].upper()

In [68]: data.index.map(transform)
Out[68]: Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

```
In [69]: data.index = data.index.map(transform)

In [70]: data
Out[70]:
one  two  three  four
OHIO    0    1      2     3
COLO    4    5      6     7
NEW     8    9     10    11
```

如果想要创建数据集的转换版（而不是修改原始数据）

```
In [71]: data.rename(index=str.title, columns=str.upper)
Out[71]:
      ONE  TWO  THREE  FOUR
Ohio    0    1      2     3
Colo    4    5      6     7
New     8    9     10    11
```

rename可以结合字典型对象实现对部分轴标签的更新

```
In [72]: data.rename(index={'OHIO': 'INDIANA'},
   ....:             columns={'three': 'peekaboo'})
Out[72]:
one  two  peekaboo  four
INDIANA    0    1         2     3
COLO       4    5         6     7
NEW        8    9        10    11
```

就地修改某个数据集，传入inplace=True即可

```
In [73]: data.rename(index={'OHIO': 'INDIANA'}, inplace=True)

In [74]: data
Out[74]:
         one  two  three  four
INDIANA    0    1      2     3
COLO       4    5      6     7
NEW        8    9     10    11
```

## 离散化和面元划分

为了便于分析，连续数据常常被离散化或拆分为"面元"（bin）。假设有一组人员数据，而你希望将它们划分为不同的年龄组

```
In [75]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

将这些数据划分为“18到25”、“26到35”、“35到60”以及“60以上”几个面元

```
In [76]: bins = [18, 25, 35, 60, 100]

In [77]: cats = pd.cut(ages, bins)

In [78]: cats
Out[78]:
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100],
(35,60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

pandas返回的是一个特殊的Categorical对象。结果展示了pandas.cut划分的面元。你可以将其看做一组表示面元名称的字符串

```
In [79]: cats.codes
Out[79]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)

In [80]: cats.categories
Out[80]:
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]]
              closed='right',
              dtype='interval[int64]')

# 面元计数
In [81]: pd.value_counts(cats)
Out[81]:
(18, 25]     5
(35, 60]     3
(25, 35]     3
(60, 100]    1
dtype: int64
```

跟“区间”的数学符号一样，圆括号表示开端，而方括号则表示闭端（包括）。哪边是闭端可以通过right=False进行修改

```
In [82]: pd.cut(ages, [18, 26, 36, 61, 100], right=False)
Out[82]:
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100),
[36,
 61), [36, 61), [26, 36)]
Length: 12
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36, 61) < [61, 100)]
```

通过传递一个列表或数组到labels，设置面元名称

```
In [83]: group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']

In [84]: pd.cut(ages, bins, labels=group_names)
Out[84]:
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged,
Mid
dleAged, YoungAdult]
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

向cut传入的是面元的数量而不是确切的面元边界，则它会根据数据的最小值和最大值计算等长面元。下面这个例子中，我们将一些均匀分布的数据分成四组，选项precision=2，限定小数只有两位

```
In [85]: data = np.random.rand(20)

In [86]: pd.cut(data, 4, precision=2)
Out[86]:
[(0.34, 0.55], (0.34, 0.55], (0.76, 0.97], (0.76, 0.97], (0.34, 0.55], ...,
(0.34
, 0.55], (0.34, 0.55], (0.55, 0.76], (0.34, 0.55], (0.12, 0.34]]
Length: 20
Categories (4, interval[float64]): [(0.12, 0.34] < (0.34, 0.55] < (0.55, 0.76]
<
(0.76, 0.97]]
```

qcut是一个非常类似于cut的函数，它可以根据样本分位数对数据进行面元划分。

```
In [87]: data = np.random.randn(1000)  # Normally distributed

In [88]: cats = pd.qcut(data, 4)  # Cut into quartiles

In [89]: cats
Out[89]:
[(-0.0265, 0.62], (0.62, 3.928], (-0.68, -0.0265], (0.62, 3.928], (-0.0265,
0.62]
, ..., (-0.68, -0.0265], (-0.68, -0.0265], (-2.95, -0.68], (0.62, 3.928],
(-0.68,
 -0.0265]]
Length: 1000
Categories (4, interval[float64]): [(-2.95, -0.68] < (-0.68, -0.0265] <
(-0.0265,
 0.62] <
                                    (0.62, 3.928]]

In [90]: pd.value_counts(cats)
Out[90]:
(0.62, 3.928]       250
(-0.0265, 0.62]     250
(-0.68, -0.0265]    250
(-2.95, -0.68]      250
dtype: int64
```

## 检测和过滤异常值

```
In [92]: data = pd.DataFrame(np.random.randn(1000, 4))

In [93]: data.describe()
Out[93]:
                 0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
```

```
mean      0.049091    0.026112   -0.002544   -0.051827
std       0.996947    1.007458    0.995232    0.998311
min      -3.645860   -3.184377   -3.745356   -3.428254
25%      -0.599807   -0.612162   -0.687373   -0.747478
50%       0.047101   -0.013609   -0.022158   -0.088274
75%       0.756646    0.695298    0.699046    0.623331
max       2.653656    3.525865    2.735527    3.366626
```

某列中绝对值大小超过3的值

```
In [94]: col = data[2]

In [95]: col[np.abs(col) > 3]
Out[95]:
41    -3.399312
136   -3.745356
Name: 2, dtype: float64
```

选出全部含有“超过3或－3的值”的行

```
In [96]: data[(np.abs(data) > 3).any(1)]
Out[96]:
            0         1         2         3
41   0.457246 -0.025907 -3.399312 -0.974657
60   1.951312  3.260383  0.963301  1.201206
136  0.508391 -0.196713 -3.745356 -1.520113
235 -0.242459 -3.056990  1.918403 -0.578828
258  0.682841  0.326045  0.425384 -3.428254
322  1.179227 -3.184377  1.369891 -1.074833
544 -3.548824  1.553205 -2.186301  1.277104
635 -0.578093  0.193299  1.397822  3.366626
782 -0.207434  3.525865  0.283070  0.544635
803 -3.645860  0.255475 -0.549574 -1.907459
```

np.sign(data)可以生成1和-1

```
In [99]: np.sign(data).head()
Out[99]:
     0    1    2    3
0 -1.0  1.0 -1.0  1.0
1  1.0 -1.0  1.0 -1.0
2  1.0  1.0  1.0 -1.0
3 -1.0 -1.0  1.0 -1.0
4 -1.0  1.0 -1.0 -1.0
```

## 排列和随机采样

利用numpy.random.permutation函数可以轻松实现对Series或DataFrame的列的排列工作（permuting，随机重排序）

```
In [100]: df = pd.DataFrame(np.arange(20).reshape((5, 4)))

In [101]: sampler = np.random.permutation(5)

In [102]: sampler
Out[102]: array([3, 1, 4, 2, 0])
```

```
In [103]: df
Out[103]:
    0   1   2   3
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
4  16  17  18  19

In [104]: df.take(sampler)
Out[104]:
    0   1   2   3
3  12  13  14  15
1   4   5   6   7
4  16  17  18  19
```

```
2   8   9  10  11
0   0   1   2   3
```

```
In [105]: df.sample(n=3)
Out[105]:
    0   1   2   3
3  12  13  14  15
4  16  17  18  19
2   8   9  10  11
```

要通过替换的方式产生样本（允许重复选择），可以传递replace=True到sample

```
In [106]: choices = pd.Series([5, 7, -1, 6, 4])

In [107]: draws = choices.sample(n=10, replace=True)

In [108]: draws
Out[108]:
4    4
1    7
4    4
2   -1
0    5
3    6
1    7
4    4
0    5
4    4
dtype: int64
```

## 计算指标/哑变量

将分类变量（categorical variable）转换为"哑变量"或"指标矩阵"

```
In [109]: df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
```

```
   .....:                      'data1': range(6)})

In [110]: pd.get_dummies(df['key'])
Out[110]:
   a  b  c
0  0  1  0
1  0  1  0
2  1  0  0
3  0  0  1
4  1  0  0
5  0  1  0
```

```
In [111]: dummies = pd.get_dummies(df['key'], prefix='key')

In [112]: df_with_dummy = df[['data1']].join(dummies)

In [113]: df_with_dummy
Out[113]:
   data1  key_a  key_b  key_c
0      0      0      1      0
1      1      0      1      0
2      2      1      0      0
3      3      0      0      1
4      4      1      0      0
5      5      0      1      0
```

```
In [114]: mnames = ['movie_id', 'title', 'genres']

In [115]: movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
   .....:                         header=None, names=mnames, engine='python')

In [116]: movies[:10]
Out[116]:
   movie_id                               title                        genres
0         1                    Toy Story (1995)   Animation|Children's|Comedy
```

```
1         2                           Jumanji (1995)  Adventure|Children's|Fantasy
2         3                  Grumpier Old Men (1995)                Comedy|Romance
3         4                 Waiting to Exhale (1995)                  Comedy|Drama
4         5       Father of the Bride Part II (1995)                        Comedy
5         6                              Heat (1995)         Action|Crime|Thriller
6         7                           Sabrina (1995)                Comedy|Romance
7         8                      Tom and Huck (1995)          Adventure|Children's
8         9                      Sudden Death (1995)
Action
9        10                         GoldenEye (1995)     Action|Adventure|Thriller
```

```
In [117]: all_genres = []


In [118]: for x in movies.genres:
   .....:     all_genres.extend(x.split('|'))


In [119]: genres = pd.unique(all_genres)
```

构建指标DataFrame的方法之一是从一个全零DataFrame开始

```
In [121]: zero_matrix = np.zeros((len(movies), len(genres)))


In [122]: dummies = pd.DataFrame(zero_matrix, columns=genres)
```

## pandas的矢量化字符串函数

```
In [167]: data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
   .....:         'Rob': 'rob@gmail.com', 'Wes': np.nan}


In [168]: data = pd.Series(data)


In [169]: data
Out[169]:
Dave     dave@google.com
```

```
Rob          rob@gmail.com
Steve      steve@gmail.com
Wes                    NaN
dtype: object

In [170]: data.isnull()
Out[170]:
Dave     False
Rob      False
Steve    False
Wes       True
dtype: bool
```

通过data.map，所有字符串和正则表达式方法都能被应用于（传入lambda表达式或其他函数）各个值，但是如果存在NA（null）就会报错。为了解决这个问题，Series有一些能够跳过NA值的面向数组方法，进行字符串操作。通过Series的str属性即可访问这些方法。例如，我们可以通过str.contains检查各个电子邮件地址是否含有"gmail"：

```
In [171]: data.str.contains('gmail')
Out[171]:
Dave     False
Rob       True
Steve     True
Wes        NaN
dtype: object
```

也可以使用正则表达式，还可以加上任意re选项（如IGNORECASE）

```
import re
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
In [172]: pattern
Out[172]: '([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\\.([A-Z]{2,4})'

In [173]: data.str.findall(pattern, flags=re.IGNORECASE)
Out[173]:
Dave     [(dave, google, com)]
Rob        [(rob, gmail, com)]
```

```
Steve    [(steve, gmail, com)]
Wes                         NaN
dtype: object
```

```
In [174]: matches = data.str.match(pattern, flags=re.IGNORECASE)

In [175]: matches
Out[175]:
Dave     True
Rob      True
Steve    True
Wes       NaN
dtype: object
```

字符串进行截取

```
In [178]: data.str[:5]
Out[178]:
Dave     dave@
Rob      rob@g
Steve    steve
Wes        NaN
dtype: object
```