

原文链接: <https://www.dataquest.io/blog/settingwithcopywarning/>

原文标题: *Understanding SettingWithCopyWarning in pandas*

原文发布时间: 5 JULY 2017 (需要注意时效性, 文中有一些方法已经弃用, 比如 `ix`)

作者: [Benjamin Pryke](#)

译者: Ivy Lee

学习 Python 数据分析的同学总是遇到这个警告, 查询中文资料, 一般只能找到个别的解决办法, 不一定适用于自己遇到的情况。查到的最常见解决办法就是直接设置为不显示警告。这实际上并不能解决问题, 搜索资料发现这篇英文讲解 `SettingWithCopyWarning` 原理非常系统的文章, 翻译了一下, 分享给大家。

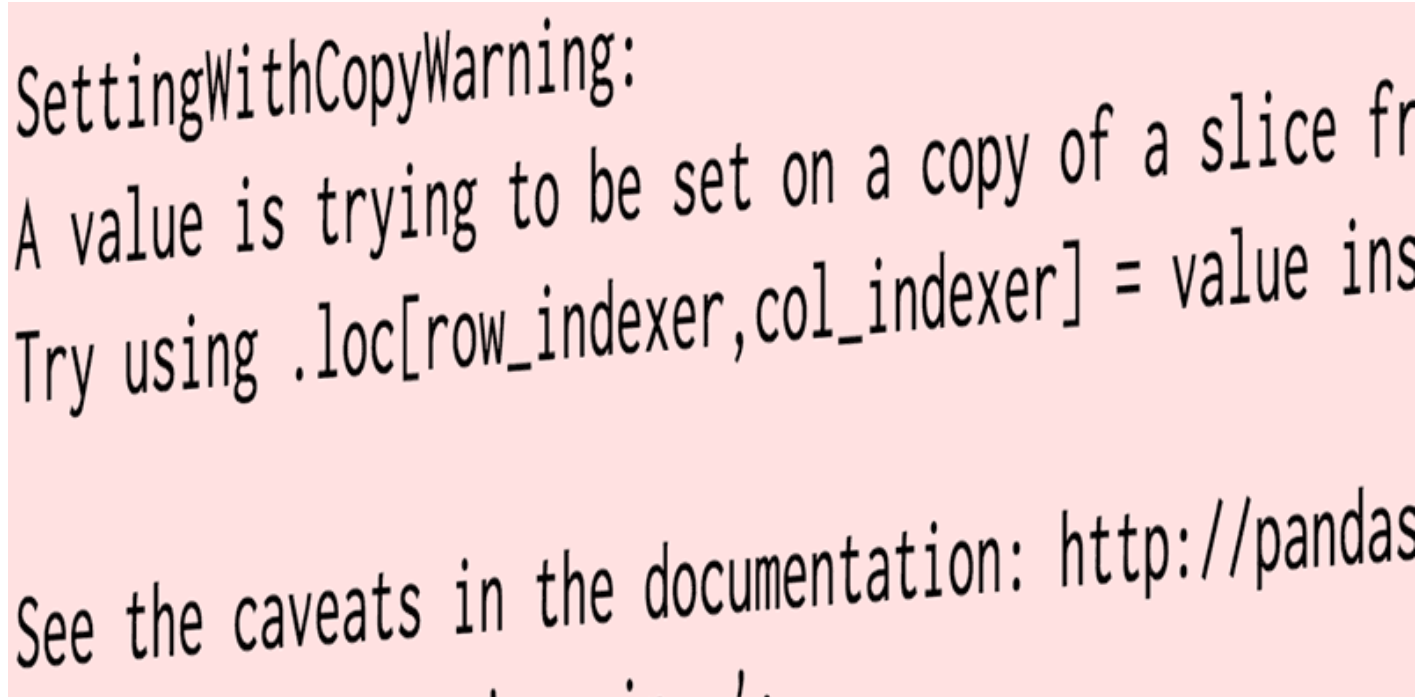
## 太长不看

- 解决方案: 学会识别链式索引, 不惜一切代价避免使用链式索引  
**注意: 如果你看不懂这里的解决方案, 请阅读此文前半部分, 直到真正理解如何去做**
  - 如果要更改原始数据, 请使用单一赋值操作 (`loc`):

```
data.loc[data.bidder == 'parakeet2004', 'bidderrate'] = 100
```
  - 如果想要一个副本, 请确保强制让 Pandas 创建副本:

```
winners = data.loc[data.bid == data.price].copy()
winners.loc[304, 'bidder'] = 'therealname'
```
- 强烈不推荐直接关闭警告, 不过还是提供一下关闭警告的设置方法:

```
pd.set_option('mode.chained_assignment', None)
```
- 深度解析底层代码和历史演变 (可选阅读)



SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead.  
See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/10min.html>

`SettingWithCopyWarning` 是人们在学习 Pandas 时遇到的最常见的障碍之一。搜索引擎可以搜索到 Stack Overflow 上的问答、GitHub issues 和一些论坛帖子, 分别提供了该警告在某些特定情况下的含义。会有这么多人同样遇到这个警告并不奇怪: 有很多方法可以索引 Pandas [数据结构](#), 每种数据结构都有各自的细微差别, 甚至 Pandas 本身并不能保证两行代码的运行结果看起来完全相同。

本指南包含了生成警告的原因及解决方案, 其中还包括一些底层细节, 让你更好地了解代码内部的运行机制, 最后提供了有关该话题的一些历史情况, 解释代码底层以这样的方式运行的原因。

为了探索 `SettingWithCopyWarning`, 我们将使用 eBay 3 天拍卖出售的 Xbox 的价格数据集, 该数据集出自 [Modelling Online Auctions](#) 一书。先来了解下数据的基本结构:

```
import Pandas as pd
```

```
data = pd.read_csv('xbox-3-day-auctions.csv')
data.head()
```

|   | auctionid  | bid   | bidtime  | bidder         | bidderrate | openbid | price |
|---|------------|-------|----------|----------------|------------|---------|-------|
| 0 | 8213034705 | 95.0  | 2.927373 | jake7870       | 0          | 95.0    | 117.5 |
| 1 | 8213034705 | 115.0 | 2.943484 | davidbresler2  | 1          | 95.0    | 117.5 |
| 2 | 8213034705 | 100.0 | 2.951285 | gladimacowgirl | 58         | 95.0    | 117.5 |
| 3 | 8213034705 | 117.5 | 2.998947 | daysrus        | 10         | 95.0    | 117.5 |
| 4 | 8213060420 | 2.0   | 0.065266 | donnie4814     | 5          | 1.0     | 120.0 |

如你所见，数据集的每一行都是某一次 eBay Xbox 出价信息。下面是对数据集中每列的简要说明：

- auctionid - 每次拍卖的唯一标识符
- bid - 本次拍卖出价
- bidtime - 拍卖的时长，以天为单位，从投标开始累计
- bidder - 投标人的 eBay 用户名
- bidderrate - 投标人的 eBay 用户评级
- openbid - 卖方为拍卖设定的开标价
- price - 拍卖结束时的中标价

## 什么是 SettingWithCopyWarning?

首先要理解的是，SettingWithCopyWarning 是一个警告 Warning，而不是错误 Error。

错误表明某些内容是“坏掉”的，例如无效语法 (invalid syntax) 或尝试引用未定义的变量；警告的作用是提醒编程人员，他们的代码可能存在潜在的错误或问题，但是这些操作在该编程语言中依然合法。在这种情况下，警告很可能表明一个严重但不容易意识到的错误。

SettingWithCopyWarning 告诉你，你的操作可能没有按预期运行，需要检查结果以确保没有出错。

如果代码确实按预期工作，那么我们会很容易忽略该警告，但是 SettingWithCopyWarning **不应该**被忽略。在进行下一步操作之前，我们需要花点时间了解这一警告显示的原因。

要了解 SettingWithCopyWarning，首先要知道，Pandas 中的某些操作会返回数据的视图 (View)，某些操作会返回数据的副本 (Copy)。

### View

|   | A | B |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |
| 2 | 5 | 6 |

df1

←df2

### Copy

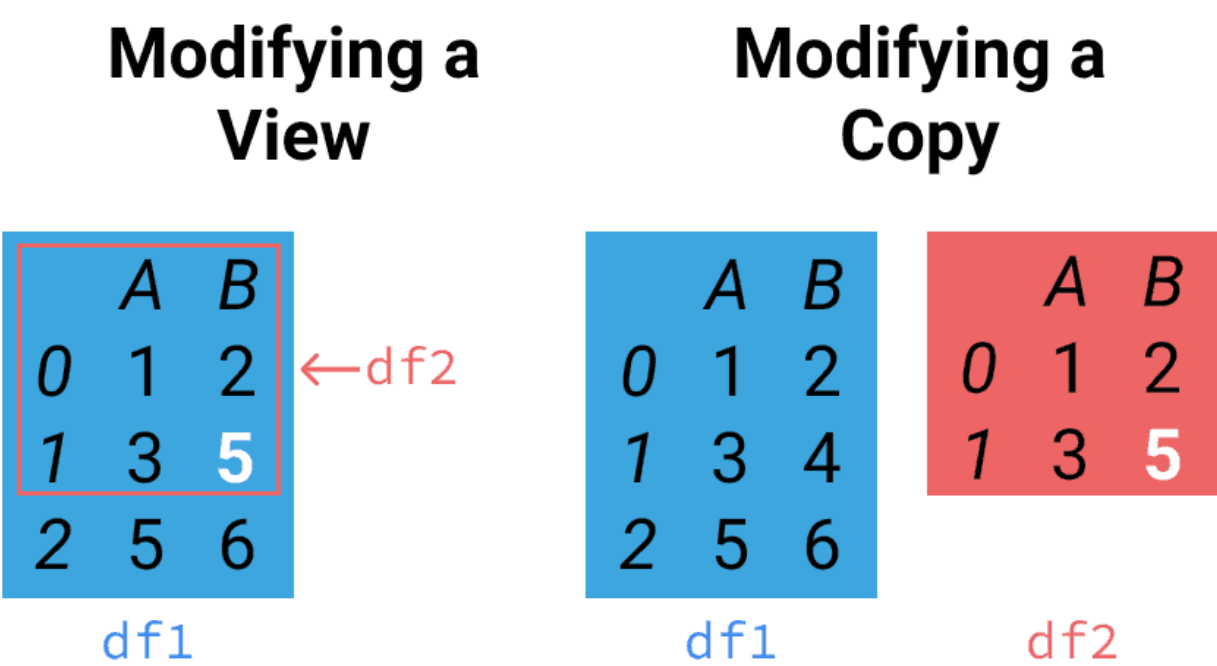
|   | A | B |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |
| 2 | 5 | 6 |

df1

|   | A | B |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |

df2

如上所示，左侧的视图 df2 只是原始数据 df1 一个子集，而右侧的副本创建了一个新的对象 df2。当我们尝试对数据集进行更改时，这可能会出现问题：



根据需求，我们可能想要修改原始 df1（左），也可能想要修改 df2（右）。警告让我们知道，代码可能并没有符合需求，修改到的可能并不是我们想要修改的那个数据集。稍后会深入研究这个问题，但是现在先来了解一下，警告出现的两个主要原因以及对应的解决方案。

## 链式赋值（Chained Assignment）

当 Pandas 检测到链式赋值（Chained Assignment）时会生成警告。为了方便后续的解释，先来解释一些术语：

- 赋值（Assignment） - 设置某些变量值的操作，例如 `data = pd.read_csv('xbox-3-day-auctions.csv')`，有时会将这个操作称之为 **设置 (Set)**。
- 访问（Access） - 返回某些值的操作，具体参照下方的索引和链式索引示例。有时会将这个操作称之为 **获取 (Get)**。
- 索引（Indexing） - 任何引用数据子集的赋值或访问方法，例如 `data[1:5]`。
- 链式索引（Chaining） - 连续使用多个索引操作，例如 `data[1:5][1:3]`。

链式赋值是链式索引和赋值的组合。先快速浏览一下之前加载的数据集，稍后将详细介绍。在这个例子中，假设我们了解到用户 'parakeet2004' 的 bidderrate 值不正确，需要修改这个 bidderrate 值，那么先来查看一下用户 'parakeet2004' 的当前值：

```
data[data.bidder == 'parakeet2004']
```

|   | auctionid  | bid   | bidtime  | bidder       | bidderrate | openbid | price |
|---|------------|-------|----------|--------------|------------|---------|-------|
| 6 | 8213060420 | 3.00  | 0.186539 | parakeet2004 | 5          | 1.0     | 120.0 |
| 7 | 8213060420 | 10.00 | 0.186690 | parakeet2004 | 5          | 1.0     | 120.0 |
| 8 | 8213060420 | 24.99 | 0.187049 | parakeet2004 | 5          | 1.0     | 120.0 |

有三行数据需要更新 bidderrate 字段，继续操作：

```
data[data.bidder == 'parakeet2004']['bidderrate'] = 100
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/ipykernel/__main__.py:1:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation:http://Pandas.pydata.org/Pandas-docs/stable/indexi
if __name__ == '__main__':
```



神奇！我们“创造”出了 SettingWithCopyWarning！  
检查一下用户 'parakeet2004' 的相关值，可以看到值没有按预期改变：  
`data[data.bidder == 'parakeet2004']`

|   | auctionid  | bid   | bidtime  | bidder       | bidderrate | openbid | price |
|---|------------|-------|----------|--------------|------------|---------|-------|
| 6 | 8213060420 | 3.00  | 0.186539 | parakeet2004 | 5          | 1.0     | 120.0 |
| 7 | 8213060420 | 10.00 | 0.186690 | parakeet2004 | 5          | 1.0     | 120.0 |
| 8 | 8213060420 | 24.99 | 0.187049 | parakeet2004 | 5          | 1.0     | 120.0 |

这次警告是因为将两个索引操作链接在一起，直接使用了两次方括号的链式索引比较容易理解。但如果使用其他访问方法，例如 `.bidderrate`、`.loc[]`、`.iloc[]`、`.ix[]`，也会如此，这次的链式操作有：

- `data[data.bidder == 'parakeet2004']`
- `['bidderrate'] = 100`

以上两个链式操作一个接一个地独立执行。第一次链式操作是为了 Get，返回一个 DataFrame，其中包含所有 bidder 等于 'parakeet2004' 的行；第二次链式操作是为了 Set，是在这个新返回的 DataFrame 上运行的，并没有修改原始的 DataFrame。  
这种情况对应的解决方案很简单：使用 `loc` 将两次链式操作组合成一步操作，确保 Pandas 进行 Set 的是原始 DataFrame。Pandas 始终确保下面这样的非链式 Set 操作起作用：

```
# 设置新值
data.loc[data.bidder == 'parakeet2004', 'bidderrate'] = 100
# 检查结果
data[data.bidder == 'parakeet2004']['bidderrate']
```

```
6      100
7      100
8      100
Name: bidderrate, dtype: int64
```

这就是警告的文本（Try using `.loc[row_indexer,col_indexer] = value` instead）中建议的操作，在这种情况下完美适用。

## 隐蔽的链式操作（Hidden chaining）

现在来看遇到 SettingWithCopyWarning 的第二种常见方式。创建一个新的 DataFrame 来探索中标者数据，因为现在已经学习了链式赋值的内容，请注意使用 `loc`：

```
winners = data.loc[data.bid == data.price]
winners.head()
```

|    | auctionid  | bid   | bidtime  | bidder             | bidderrate | openbid | price |
|----|------------|-------|----------|--------------------|------------|---------|-------|
| 3  | 8213034705 | 117.5 | 2.998947 | daysrus            | 10         | 95.00   | 117.5 |
| 25 | 8213060420 | 120.0 | 2.999722 | djnoeproductions   | 17         | 1.00    | 120.0 |
| 44 | 8213067838 | 132.5 | 2.996632 | *champaignbubbles* | 202        | 29.99   | 132.5 |
| 45 | 8213067838 | 132.5 | 2.997789 | *champaignbubbles* | 202        | 29.99   | 132.5 |

|    | auctionid  | bid   | bidtime  | bidder  | bidderrate | openbid | price |
|----|------------|-------|----------|---------|------------|---------|-------|
| 66 | 8213073509 | 114.5 | 2.999236 | rr6kids | 4          | 1.00    | 114.5 |

winners 变量可能会被用来编写一些后续代码：

```
mean_win_time = winners.bidtime.mean()
... # 20 lines of code
mode_open_bid = winners.openbid.mode()
```

我们在偶然间发现了一个数据错误：标记为 304 的行中缺少了 bidder 值。

```
winners.loc[304, 'bidder']
```

```
nan
```

对这个例子来说，假设我们已知该投标人的真实用户名，并据此更新数据：

```
winners.loc[304, 'bidder'] = 'therealname'
```

```
/Library/Frameworks/Python.framework/Versions/36/lib/python3.6/Pandas/core/indexing.py:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://Pandas.pydata.org/Pandas-docs/stable/index
self.obj[item] = s
```

另一个 SettingWithCopyWarning! 但是这次使用了 loc，为什么还会出现警告？让我们来看代码的结果来一探究竟：

```
print(winners.loc[304, 'bidder'])
```

```
therealname
```

代码确实起了预期的作用，为什么仍然出现警告？

链式索引可能在一行代码内发生，也可能跨越两行代码。因为 winners 变量是作为 Get 操作的输出创建的（data.loc[data.bid == data.price]），它可能是原始 DataFrame 的副本，也可能不是，除非检查，否则我们不能确认。对 winners 进行索引时，实际上使用的就是链式索引。

这意味着当我们尝试修改 winners 时，可能也修改了 data。

在实际的代码中，相关的两行链式索引代码之间，可能相距很多行其他代码，追踪问题可能会更困难，但大致情况是与示例类似的。

这种情况下的警告解决方案是：创建新 DataFrame 时明确告知 Pandas 创建一个副本：

```
winners = data.loc[data.bid == data.price].copy()
winners.loc[304, 'bidder'] = 'therealname'
print(winners.loc[304, 'bidder'])
print(data.loc[304, 'bidder'])
```

```
therealname
nan
```

就这么简单！

窍门就是，学会识别链式索引，不惜一切代价避免使用链式索引。如果要更改原始数据，请使用单一赋值操作。如果你想要一个副本，请确保你强制让 Pandas 创建副本。这样既可以节省时间，也可以使代码保持逻辑严密。

另外请注意，即使 SettingWithCopyWarning 只在你进行 Set 时才会发生，但在进行 Get 操作时，最好也避免使用链式索引。链式操作代码效率较低，而且只要稍后进行赋值，就会导致问题。

## 处理 SettingWithCopyWarning 的提示和技巧

在进行下面更深入的分析之前，让我们看看 `SettingWithCopyWarning` 的更多细节。

## 关闭警告

如果不讨论如何明确地控制 `SettingWithCopy` 警告设置，本文则不够完整。Pandas 的 `mode.chained_assignment` 选项可以采用以下几个值之一：

- 'raise' - 抛出异常 (exception) 而不是警告
- 'warn' - 生成警告 (默认)
- None - 完全关闭警告

例如，如果要关闭警告：

```
pd.set_option('mode.chained_assignment', None)
data[data.bidder == 'parakeet2004']['bidderrate'] = 100
```

这样没有给出任何提示或警告，除非完全了解代码的运行情况，否则请不要尝试。只要你对想要实现的代码功能有任何一丁点疑问，不要关闭警告。有些开发者非常重视 `SettingWithCopy` 甚至选择将其提升为异常，如下所示：

```
pd.set_option('mode.chained_assignment', 'raise')
data[data.bidder == 'parakeet2004']['bidderrate'] = 100

-----
SettingWithCopyError                                Traceback (most recent call last)
<ipython-input-13-80e3669cab86> in <module>()
      1 pd.set_option('mode.chained_assignment', 'raise')
----> 2 data[data.bidder == 'parakeet2004']['bidderrate'] = 100

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/Pandas/core/frame.py in
  2427         else:
  2428             # set column
-> 2429         self._set_item(key, value)
  2430
  2431     def _setitem_slice(self, key, value):

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/Pandas/core/frame.py in
  2500         # value exception to occur first
  2501         if len(self):
-> 2502             self._check_setitem_copy()
  2503
  2504     def insert(self, loc, column, value, allow_duplicates=False):

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/Pandas/core/generic.py
  1758
  1759         if value == 'raise':
-> 1760             raise SettingWithCopyError(t)
  1761         elif value == 'warn':
  1762             warnings.warn(t, SettingWithCopyWarning, stacklevel=stacklevel)
```

`SettingWithCopyError`:

A value is trying to be `set` on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://Pandas.pydata.org/Pandas-docs/stable/index>

如果你正与缺乏经验的 Pandas 开发人员合作开发项目，或者正在开发需要高度严谨的项目，这可能特别有用。  
更精确使用此设置的方法是使用 [上下文管理器 context manager](#)。

```
# resets the option we set in the previous code segment
pd.reset_option('mode.chained_assignment')
```



```
with pd.option_context('mode.chained_assignment', None):
    data[data.bidder == 'parakeet2004']['bidderrate'] = 100
```

如你所见，这种方法可以实现针对性的警告设置，而不影响整个环境。

## is\_copy 属性

避免警告的另一个技巧是修改 Pandas 用于解释 SettingWithCopy 的工具之一。每个 DataFrame 都有一个 is\_copy 属性，默认情况下为 None，但如果它是副本，则会使用 weakref 引用原始 DataFrame。通过将 is\_copy 设置为 None，可以避免生成警告。

```
winners = data.loc[data.bid == data.price]
winners.is_copy = None
winners.loc[304, 'bidder'] = 'therealname'
```

但是请注意，这**并不会**奇迹般地解决问题，反而会使错误检测变得更加困难。

## 单类型 VS 多类型对象

值得强调的另一点是单类型对象和多类型对象之间的差异。如果 DataFrame 所有列都具有相同的 dtype，则它是单类型的，例如：

```
import numpy as np

single_dtype_df = pd.DataFrame(np.random.rand(5,2), columns=list('AB'))
print(single_dtype_df.dtypes)
single_dtype_df

A    float64
B    float64
dtype: object
```

|   | A        | B        |
|---|----------|----------|
| 0 | 0.383197 | 0.895652 |
| 1 | 0.077943 | 0.905245 |
| 2 | 0.452151 | 0.677482 |
| 3 | 0.533288 | 0.768252 |
| 4 | 0.389799 | 0.674594 |

如果 DataFrame 的列不是全部具有相同的 dtype，那么它是多类型的，例如：

```
multiple_dtype_df = pd.DataFrame({'A': np.random.rand(5), 'B': list('abcde')})
print(multiple_dtype_df.dtypes)
multiple_dtype_df

A    float64
B    object
dtype: object
```

|   | A        | B |
|---|----------|---|
| 0 | 0.615487 | a |
| 1 | 0.946149 | b |
| 2 | 0.701231 | c |
| 3 | 0.756522 | d |
| 4 | 0.481719 | e |

由于下面**历史**部分中所述的原因，对多类型对象的索引 Get 操作将始终返回副本。而为了提高效率，索引器对单类型对象的操作几乎总是返回一个视图，需要注意的是，这取决于对象的内存布局，并不能完全保证。

## 误报

误报，即无意中报告链式赋值的情况，曾经在早期版本的 Pandas 中比较常见，但此后大部分都被解决了。为了完整起见，在本文中包含一些已修复的误报示例也是有用的。如果你在使用早期版本的 Pandas 时遇到以下任何情况，则可以安全地忽略或抑制警告（或通过升级 Pandas 版本完全避免警告！）

使用当前列的值，将新列添加到 DataFrame 会**生成警告**，但这已得到修复。

```
data['bidtime_hours'] = data.bidtime.map(lambda x: x * 24)
data.head(2)
```

|   | auctionid  | bid   | bidtime  | bidder        | bidderrate | openbid | price | bidtime_hours |
|---|------------|-------|----------|---------------|------------|---------|-------|---------------|
| 0 | 8213034705 | 95.0  | 2.927373 | jake7870      | 0          | 95.0    | 117.5 | 70.256952     |
| 1 | 8213034705 | 115.0 | 2.943484 | davidbresler2 | 1          | 95.0    | 117.5 | 70.643616     |

在一个 DataFrame 切片上使用 apply 方法进行 Set 时，**也会出现误报**，不过这也已得到修复。

```
data.loc[:, 'bidtime_hours'] = data.bidtime.apply(lambda x: x * 24)
data.head(2)
```

|   | auctionid  | bid   | bidtime  | bidder        | bidderrate | openbid | price | bidtime_hours |
|---|------------|-------|----------|---------------|------------|---------|-------|---------------|
| 0 | 8213034705 | 95.0  | 2.927373 | jake7870      | 0          | 95.0    | 117.5 | 70.256952     |
| 1 | 8213034705 | 115.0 | 2.943484 | davidbresler2 | 1          | 95.0    | 117.5 | 70.643616     |

直到 0.17.0 版本前，DataFrame.sample 方法中存在一个错误，导致 SettingWithCopy 警告误报。现在，sample 方法每次都会返回一个副本。

```
sample = data.sample(2)
sample.loc[:, 'price'] = 120
sample.head()
```

|     | auctionid  | bid    | bidtime  | bidder     | bidderrate | openbid | price | bidtime_hours |
|-----|------------|--------|----------|------------|------------|---------|-------|---------------|
| 481 | 8215408023 | 91.01  | 2.990741 | sailer4eva | 1          | 0.99    | 120   | 71.777784     |
| 503 | 8215571039 | 100.00 | 1.965463 | lambonius1 | 0          | 50.00   | 120   | 47.171112     |

## 链式赋值深度解析

让我们重用之前的例子：试图更新 data 中 bidder 值为 'parakeet2004' 的所有行的 bidderrate 字段。

```
data[data.bidder == 'parakeet2004']['bidderrate'] = 100
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/ipykernel/__main__.py:1
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: http://Pandas.pydata.org/Pandas-docs/stable/index
if __name__ == '__main__':
```

Pandas 用 SettingWithCopyWarning 告诉我们的是，代码的行为是模棱两可的，要理解原因和警告的措辞，以下概念将会有所帮助。

之前简要了解了视图 (View) 和副本 (Copy)。有两种方法可以访问 DataFrame 的子集：可以创建对内存中原始数据的引用 (视图)，也可以将子集复制到新的较小的 DataFrame 中 (副本)。视图是查看 **原始** 数据特定部分的



一种方式；副本是将该数据 **复制** 到内存中的新位置。正如之前的图表所示，修改视图将修改原始变量，而修改副本则不会。

由于某些原因（本文稍后介绍），Pandas 中 Get 操作的输出无法保证。索引 Pandas 数据结构时，视图或副本都可能被返回，也就是说：对某一 DataFrame 进行 Get 操作返回一个新的 DataFrame，新的数据可能是：

- 来自原始对象的数据副本
- 没有复制，而是直接对原始对象的引用

因为不确定返回的对象是什么，而且每种可能性都有非常不同后续影响，所以忽略警告就是“玩火”。

为了更清楚地解释视图、副本和其中的歧义，我们创建一个简单的 DataFrame 并对其进行索引：

```
df1 = pd.DataFrame(np.arange(6).reshape((3,2)), columns=list('AB'))
df1
```

|   | A | B |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 2 | 3 |
| 2 | 4 | 5 |

将 df1 的子集赋值给 df2：

```
df2 = df1.loc[:1]
df2
```

|   | A | B |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 2 | 3 |

根据刚才学到的知识，我们知道 df2 可能是 df1 的视图或 df1 子集的副本。

在解决问题之前，我们还需要再看一下链式索引。扩展一下 'parakeet2004' 示例，将两个索引操作链接在一起：

```
data[data.bidder == 'parakeet2004']
__intermediate__['bidderrate'] = 100
```

`__intermediate__` 表示第一个调用的输出，对我们是完全不可见的。请记住，如果我们使用了属性访问（`.+`列名形式的访问），会得到相同的有问题的结果：

```
data[data.bidder == 'parakeet2004'].bidderrate = 100
```

这同样适用于任何其他形式的链式调用，**因为我们正在生成中间对象**。

在底层代码中，链式索引意味着对 `__getitem__` 或 `__setitem__` 进行多次调用以完成单个操作。这些是 **特殊的 Python 方法**，通过在实现它们类的实例上使用方括号，可以调用这些方法，这是一种 **语法糖**。下面看一下 Python 解释器如何执行示例中的内容。

```
# Our code
```

```
data[data.bidder == 'parakeet2004']['bidderrate'] = 100
```

```
# Code executed
```

```
data.__getitem__(data.__getitem__('bidder') == 'parakeet2004').__setitem__('bidderrate'
```

你可能已经意识到，`SettingWithCopyWarning` 是由此链式 `__setitem__` 调用生成的。可以自己尝试一下 - 上面这些代码的功能相同。为清楚起见，请注意第二个 `__getitem__` 调用（对 `bidder` 列）是嵌套的，而不是链式问题的所有部分。

通常，如上面所述，Pandas 不保证 Get 操作是返回视图还是副本。如果示例中返回了一个视图，则链式赋值中的第二个表达式将是对原始对象 `__setitem__` 的调用。但是，如果返回一个副本，那么将被修改的是副本 - 原始对象不会被修改。

这就是警告中“a value is trying to be set on a copy of a slice from a DataFrame”的含义。由于没有对此副本的引用，它最终将被回收。SettingWithCopyWarning 让我们知道 Pandas 无法确定第一个 \_\_getitem\_\_ 调用是否返回了视图或副本，因此不清楚该赋值是否更改了原始对象。换一种说法就是：“我们是否正在修改原始数据？”这一问题的答案是未知的。

如果确实想要修改原始文件，警告建议的解决方案是使用 loc 将这两个单独的链式操作转换为单个赋值操作。这样代码中没有了链式索引，就不会再收到警告。修改后的代码及其扩展版本如下所示：

```
# Our code
data.loc[data.bidder == 'parakeet2004', 'bidderrate'] = 100

# Code executed
data.loc.__setitem__((data.__getitem__('bidder') == 'parakeet2004', 'bidderrate'), 100)
```

DataFrame 的 loc 属性保证是原始 DataFrame 本身，具有扩展的索引功能。

## 假阴性 (False negatives)

使用 loc 并没有结束问题，因为使用 loc 的 Get 操作仍然可以返回一个视图或副本，下面是个有点复杂的例子。

```
data.loc[data.bidder == 'parakeet2004', ('bidderrate', 'bid')]
```

|   | bidderrate | bid   |
|---|------------|-------|
| 6 | 100        | 3.00  |
| 7 | 100        | 10.00 |
| 8 | 100        | 24.99 |

这次拉出了两列而不是一列。下面尝试 Set 所有的 bid 值。

```
data.loc[data.bidder == 'parakeet2004', ('bidderrate', 'bid')]['bid'] = 5.0
data.loc[data.bidder == 'parakeet2004', ('bidderrate', 'bid')]
```

|   | bidderrate | bid   |
|---|------------|-------|
| 6 | 100        | 3.00  |
| 7 | 100        | 10.00 |
| 8 | 100        | 24.99 |

没有效果，也没有警告！我们在切片的副本上 Set 了一个值，但是 Pandas 没有检测到它 - 这就是假阴性。这是因为，使用 loc 之后并不意味着可以再次使用链式赋值。这个特定的 bug，有一个未解决的 [GitHub issue](#)。

正确的解决方法如下：

```
data.loc[data.bidder == 'parakeet2004', 'bid'] = 5.0
data.loc[data.bidder == 'parakeet2004', ('bidderrate', 'bid')]
```

|   | bidderrate | bid |
|---|------------|-----|
| 6 | 100        | 5   |
| 7 | 100        | 5   |
| 8 | 100        | 5   |

你可能怀疑，是否真的有人会在实践中遇到这样的问题。其实这比你想象的更容易出现。当我们像下一节中这样做：将 DataFrame 查询的结果赋值给变量。

## 隐藏的链式索引

再看一下之前隐藏的链式索引示例，我们试图设置 winners 变量中，标记为 304 行的 bidder 字段。

```
winners = data.loc[data.bid == data.price]
winners.loc[304, 'bidder'] = 'therealname'
```

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/Pandas/core/indexing.py  
A value is trying to be **set** on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://Pandas.pydata.org/Pandas-docs/stable/index>  
self.obj[item] = s

尽管使用了 `loc`，还是得到了 `SettingWithCopyWarning`。这可能令人非常困惑，因为警告信息建议的方法，我们已经做过了。

不过，想一下 `winners` 变量究竟是什么？由于我们通过 `data.loc[data.bid == data.price]` 将它初始化，无法知道它是原始 `data` 的视图还是副本（因为 `Get` 操作返回视图或副本）。将初始化与生成警告的行组合在一起可以清楚地表明我们的错误。

```
data.loc[data.bid == data.price].loc[304, 'bidder'] = 'therealname'
```

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/Pandas/core/indexing.py  
A value is trying to be **set** on a copy of a slice from a DataFrame.  
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: <http://Pandas.pydata.org/Pandas-docs/stable/index>  
self.obj[item] = s

再次使用了链式赋值，只是这次它被分在了两行代码中。思考这个问题的另一种方法是，问一个问题：“这个操作会修改一个对象，还是两个对象？”在示例中，答案是未知的：如果 `winners` 是副本，那么只有 `winners` 受到影响，但如果是视图，则 `winners` 和 `data` 都将被更新。这种情况可能发生在脚本或代码库中相距很远的行之间，这使问题很难被追根溯源。

此处警告的意图是提醒，自以为代码将修改原始 `DataFrame`，实际没有修改成功，或者说我们将修改副本而不是原始数据。深入研究 [Pandas GitHub repo](#) 中的 [issue](#)，可以看到开发人员自己对这个问题的解释。

如何解决这个问题在很大程度上取决于自己的意图。如果想要使用原始数据的副本，解决方案就是强制 `Pandas` 制作副本。

```
winners = data.loc[data.bid == data.price].copy()
winners.loc[304, 'bidder'] = 'therealname'
```

```
print(data.loc[304, 'bidder']) # Original
print(winners.loc[304, 'bidder']) # Copy
```

```
nan
therealname
```

另一方面，如果需要更新原始 `DataFrame`，那么应该使用原始 `DataFrame` 而不是重新赋值一些具有未知行为的其他变量。之前的代码可以修改为：

```
# Finding the winners
winner_mask = data.bid == data.price
```

```
# Taking a peek
data.loc[winner_mask].head()
```

```
# Doing analysis
mean_win_time = data.loc[winner_mask, 'bidtime'].mean()
... # 20 lines of code
mode_open_bid = data.loc[winner_mask, 'openbid'].mode()
```

```
# Updating the username
data.loc[304, 'bidder'] = 'therealname'
```

在更复杂的情况下，例如修改 DataFrame 子集的子集，不要使用链式索引，可以在原始 DataFrame 上通过 loc 进行修改。例如，可以更改上面的新 winner\_mask 变量或创建一个选择中标者子集的新变量，如下所示：

```
high_winner_mask = winner_mask & (data.price > 150)
data.loc[high_winner_mask].head()
```

|     | auctionid  | bid   | bidtime  | bidder            | bidderrate | openbid | price | bidtime_hours |
|-----|------------|-------|----------|-------------------|------------|---------|-------|---------------|
| 225 | 8213387444 | 152.0 | 2.919757 | uconnbabydoll1975 | 15         | 0.99    | 152.0 | 70.074168     |
| 328 | 8213935134 | 207.5 | 2.983542 | toby2492          | 0          | 0.10    | 207.5 | 71.605008     |
| 416 | 8214430396 | 199.0 | 2.990463 | volpendesta       | 4          | 9.99    | 199.0 | 71.771112     |
| 531 | 8215582227 | 152.5 | 2.999664 | ultimatum_man     | 2          | 60.00   | 152.5 | 71.991936     |

这种技术会使未来的代码库维护和扩展地更加稳健。

## 历史

你可能想知道为什么要造成这么混乱的现状，为什么不明确指定索引方法是返回视图还是副本，来完全避免 SettingWithCopy 问题。要理解这个问题，必须研究 Pandas 的过去。

Pandas 确定返回一个视图还是一个副本的逻辑，源于它对 NumPy 库的使用，这是 Pandas 库的基础。视图实际上是通过 NumPy 进入 Pandas 的词库的。实际上，视图在 NumPy 中很有用，因为它们能够可预测地返回。由于 NumPy 数组是单一类型的，因此 Pandas 尝试使用最合适的 dtype 来最小化内存处理需求。因此，包含单个 dtype 的 DataFrame 切片可以作为单个 NumPy 数组的视图返回，这是一种高效处理方法。但是，多类型的切片不能以相同的方式存储在 NumPy 中。Pandas 兼顾多种索引功能，并且保持高效地使用其 NumPy 内核的能力。

最终，Pandas 中的索引被设计为有用且通用的方式，其核心并不完全与底层 NumPy 数组的功能相结合。随着时间的推移，这些设计和功能元素之间的相互作用，导致了一组复杂的规则，这些规则决定了返回视图还是副本。经验丰富的 Pandas 开发者通常都很满意 Pandas 的做法，因为他们可以轻松地浏览其索引行为。

不幸的是，对于 Pandas 的新手来说，链式索引几乎不可避免，因为 Get 操作返回的就是可索引的 Pandas 对象。此外，用 Pandas 的核心开发人员之一 [Jeff Reback](#) 的话来说，“从语言的角度来看，直接检测链式索引是不可能的，必须经过推断才能了解”（It is simply not possible from a language perspective to detect chain indexing directly; it has to be inferred）。

因此，在 2013 年底的 0.13.0 版本中引入了警告，作为许多开发者遇到链式赋值导致的无声失败的解决方案。

在 0.12 版本之前，ix 索引器是最受欢迎的（在 Pandas 术语中，“索引器”比如 ix, loc 和 iloc，是一种简单的结构，允许使用方括号来索引对象，就像数组一样，但具有一些特殊的用法）。但是大约在 2013 年中，Pandas 项目开始意识到日益增加的新手用户的重要性，有动力开始提高新手用户的使用体验。自从此版本发布以来，loc 和 iloc 索引器因其更明确的性质和更易于解释的用法而受到青睐。（译者注：pandas v0.23.3 (July 7, 2018)，其中 ix 方法已经被弃用）



SettingWithCopyWarning 在推出后持续改进，多年来在许多 GitHub issue 中得到了热烈的讨论，甚至还在不断更新，但是要理解它，仍然是成为 Pandas 专家的关键。

## 总结

SettingWithCopyWarning 的基础复杂性是 Pandas 库中为数不多的坑。这个警告的源头深深嵌在库的底层中，不应被忽视。Jeff Reback [自己的话](#)，“There are no cases that I am aware that you should actually ignore this warning. ....If you do certain types of indexing it will never work, others it will work. You are really playing with fire.”

幸运的是，解决警告只需要**识别链式赋值并将其修复**——看完本文你唯一需要理解的事。