Pandas循环提速7万多倍！Python数据分析攻略

Python那些事 9月5日

（点击上方快速关注并设置为星标，一起学Python）

> **乾明 编译整理**
> **量子位 报道 | 公众号 QbitAI**

用Python和Pandas进行数据分析，很快就会用到循环。

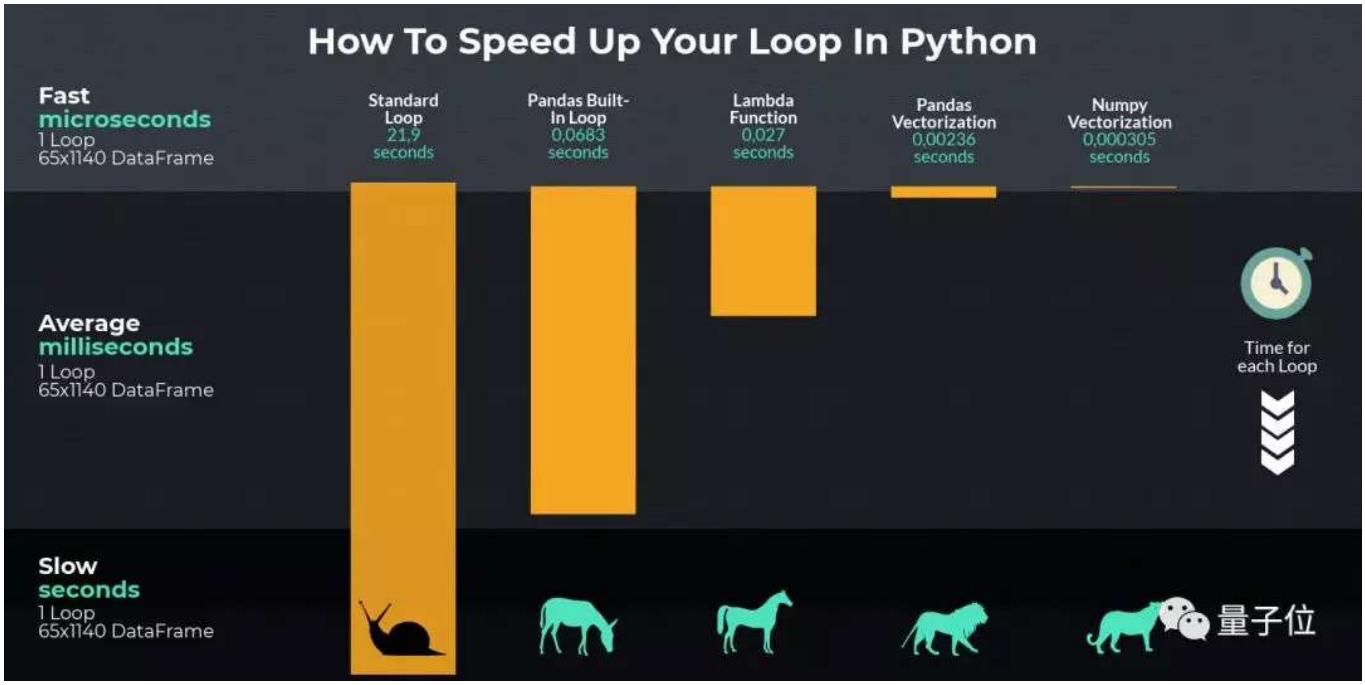
但在这其中，就算是较小的DataFrame，使用标准循环也比较耗时。

遇到较大的DataFrame时，需要的时间会更长，会让人更加头疼。

现在，有人忍不了了。他是一位来自德国的数据分析师，名叫Benedikt Droste。

他说，当自己花了大半个小时等待代码执行的时候，决定寻找速度更快的替代方案。

在给出的替代方案中，使用Numpy向量化，与使用标准循环相比，速度提升了71803倍。



他是怎么实现的？我们一起来看看~

标准循环处理3年足球赛数据：20.7秒

DataFrame是具有行和列的Pandas对象。如果使用循环，需要遍历整个对象。

Python不能利用任何内置函数，而且速度很慢。在Benedikt Droste的提供的示例中，是一个包含65列和1140行的Dataframe，包含了2016-2019赛季的足球赛结果。

需要解决的问题是：创建一个新的列，用于指示某个特定的队是否打了平局。可以这样开始：

```
def soc_loop(leaguedf,TEAM,):
    leaguedf['Draws'] = 99999
    for row in range(0, len(leaguedf)):
        if ((leaguedf['HomeTeam'].iloc[row] == TEAM) & (leaguedf['FTR'].iloc[row] == 'D'
            ((leaguedf['AwayTeam'].iloc[row] == TEAM) & (leaguedf['FTR'].iloc[row] == 'D
            leaguedf['Draws'].iloc[row] = 'Draw'
        elif ((leaguedf['HomeTeam'].iloc[row] == TEAM) & (leaguedf['FTR'].iloc[row] != '
            ((leaguedf['AwayTeam'].iloc[row] == TEAM) & (leaguedf['FTR'].iloc[row] != 'D
            leaguedf['Draws'].iloc[row] = 'No_Draw'
        else:
            leaguedf['Draws'].iloc[row] = 'No_Game'
```

```
%%timeit

soc_loop(df,'Arsenal')
```



```
20.7 s ± 355 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

在这个案例中是阿森纳，在实现目标之前要确认阿森纳参加了哪些场比赛，是主队还是客队。但使用标准循环非常慢，执行时间为20.7秒。

那么，怎么才能更有效率？

Pandas 内置函数: iterrows ()一快321倍

在第一个示例中，循环遍历了整个DataFrame。iterrows()为每一行返回一个Series，它以索引对的形式遍历DataFrame，以Series的形式遍历感兴趣的列。这使得它比标准循环更快：

```
def soc_iter(TEAM,home,away,ftr):
    #team, row['HomeTeam'], row['AwayTeam'], row['FTR']
    if [((home == TEAM) & (ftr == 'D')) | ((away == TEAM) & (ftr == 'D'))]:
        result = 'Draw'
    elif [((home == TEAM) & (ftr != 'D')) | ((away == TEAM) & (ftr != 'D'))]:
        result = 'No_Draw'
    else:
        result = 'No_Game'
    return result
```

```
%%timeit

draw_series = []
for index, row in df.iterrows():
    draw_series.append(soc_iter('Arsenal',row['HomeTeam'], row['AwayTeam'], row['FTR']))
df['Draws'] = draw_series
```



```
87.7 ms ± 14 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

代码运行时间为68毫秒，比标准循环快321倍。但是，许多人建议不要使用它，因为仍然有更快的选项，而且iterrows()不能跨行保存dtype。

这意味着，如果你在DataFrame dtypes上使用iterrows()，可以更改它，但这会导致很多问题。

一定要保存dtypes的话，你还可以使用itertuples()。这里我们不详细讨论 ，你可以在这里找到官方文件：

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.itertuples.html

apply ()方法一快811倍

apply 本身并不快，但与DataFrame结合使用时，它具有优势。这取决于 apply 表达式的内容。如果可以在 Cython 空间中执行，那么apply要快得多，这里的示例就是这种情况。

大家可以在Lambda函数中使用apply。所要做的就是指定这个轴。在本文的示例中，想要执行按列操作，要使用 axis 1：

```
%%timeit
df['Draws'] = df.apply(lambda row: soc_iter('Arsenal',row['HomeTeam'], row['AwayTeam'], row['FTR']), axis=1)
27.7 ms ± 1.94 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```


这段代码甚至比之前的方法更快，完成时间为27毫秒。

Pandas向量化—快9280倍

此外，也可以利用向量化的优点来创建非常快的代码。

重点是避免像之前的示例中的Python级循环，并使用优化后的C语言代码，这将更有效地使用内存。只需要稍微修改一下函数：

```
def soc_iter(TEAM,home,away,ftr):
    df['Draws'] = 'No_Game'
    df.loc[((home == TEAM) & (ftr == 'D')) | ((away == TEAM) & (ftr == 'D')), 'Draws'] =
    df.loc[((home == TEAM) & (ftr != 'D')) | ((away == TEAM) & (ftr != 'D')), 'Draws'] =
```

现在，可以用 Pandas 列作为输入创建新列：

```
%%timeit
df['Draws'] = soc_iter('Arsenal',df['HomeTeam'], df['AwayTeam'], df['FTR'])
1.39 ms ± 65.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

在这种情况下，甚至不需要循环。所要做的就是调整函数的内容。现可以直接将Pandas 列传递给函数，从而获得巨大的速度增益。

Numpy向量化—快71803倍

在上面的示例中，将将Pandas 列传递给函数。通过添加.values，可以得到一个Numpy数组：

```
%%timeit
df['Draws'] = soc_iter('Arsenal',df['HomeTeam'].values, df['AwayTeam'].values, df['FTR'].values)
258 µs ± 7.95 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

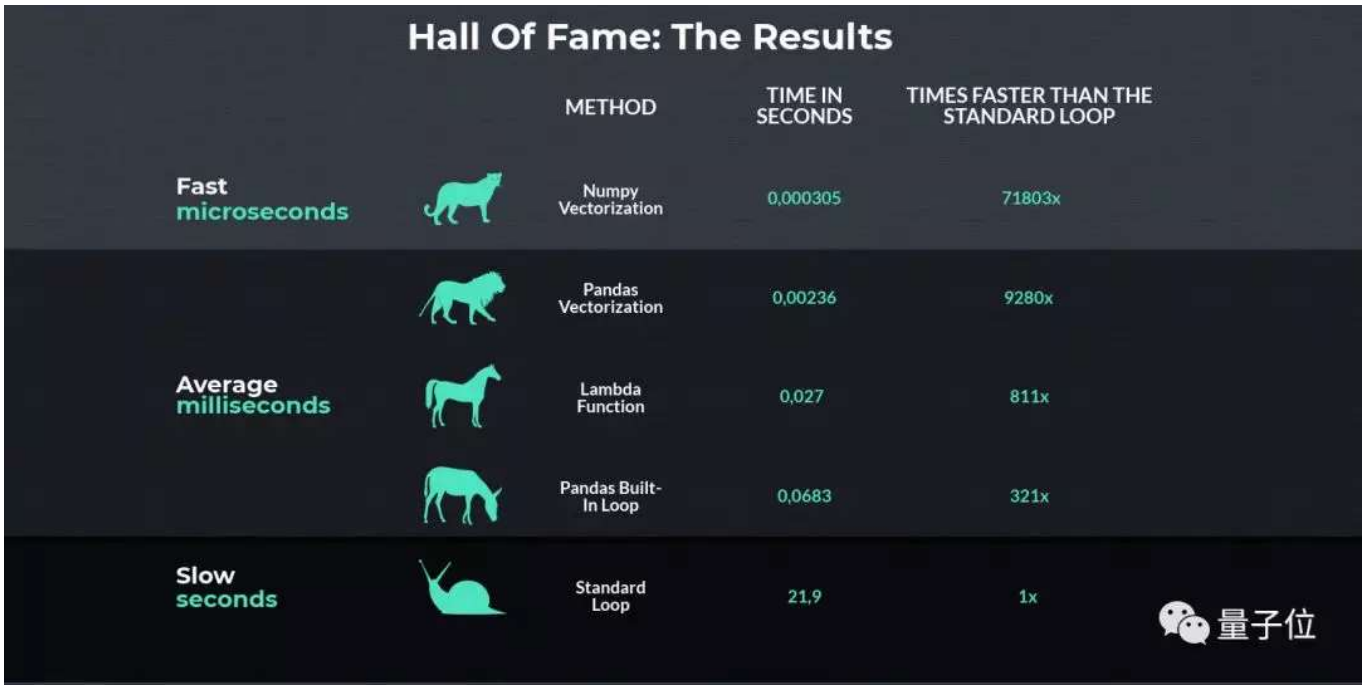

因为引用了局部性的好处，Numpy数组的速度非常快，代码运行时间仅为0.305毫秒，比一开始使用的标准循环快71803倍。

谁更强一目了然

最后，Benedikt Droste对上述方案进行了总结。

他说，如果你使用Python、Pandas和Numpy进行数据分析，总会有改进代码的空间。

在对上述五种方法进行比较之后，哪个更快一目了然：



从这个图中，可以得出两个结论：

- 1、如果要使用循环，则应始终选择apply方法。
- 2、否则，使用向量化是最好的，因为它更快！

原文链接：

https://towardsdatascience.com/how-to-make-your-pandas-loop-71-803-times-faster-805030df4f06

(完)

看完本文有收获？请转发分享给更多人

关注「Python那些事」，做全栈开发工程师

点「在看」的人都变好看了哦