

A Misconception About Pandas Inplace

The counterintuitive behaviour of inplace operations.



AVI CHAWLA

JUN 18, 2024



29



1



Share



Advertise to 79k readers / Deep Dives

Most Pandas users have a misconception about inplace operations.

They profoundly use them in expectation of:

- Smaller run-time
- Lower memory usage

And, of course, the reasoning makes intuitive sense as well.

Inplace, as the name suggests, must modify the DataFrame without creating a new copy. Thus, it is okay to expect that inplace will be more efficient.

Yet, **this is rarely the case**, which is also evident from the image below:

Don't use **Inplace** Operations in Pandas



blog.DailyDoseofDS.com

| Method | <i>inplace=False</i> | <i>inplace=True</i> | Remark |
|-----------------------------------|----------------------|---------------------|-----------------|
| <code>df.replace()</code> | 140 μ s | 244 μ s | Inplace is Slow |
| <code>df.sort_values()</code> | 374 μ s | 450 μ s | Inplace is Slow |
| <code>df.reset_index()</code> | 35 μ s | 10 μ s | Inplace is Fast |
| <code>df.drop()</code> | 200 μ s | 262 μ s | Inplace is Slow |
| <code>df.fillna()</code> | 90 μ s | 222 μ s | Inplace is Slow |
| <code>df.dropna()</code> | 750 μ s | 1088 μ s | Inplace is Slow |
| <code>df.drop_duplicates()</code> | 856 μ s | 1058 μ s | Inplace is Slow |
| <code>df.rename()</code> | 151 μ s | 152 μ s | Same speed |

It is clear that in most cases, inplace operations are slow.

Why does this happen?

Contrary to common belief, Pandas' inplace operations **NEVER** prevent the creation of a new copy.

It is just that these operations assign the copy back to the same address.

But during this assignment step, Pandas has to perform some additional checks — `SettingWithCopy`, for instance, to ensure that the DataFrame is being modified correctly.

This, at times, can be an expensive operation.

Yet, in general, there is no guarantee that an inplace operation is faster, which is also validated by the results above.

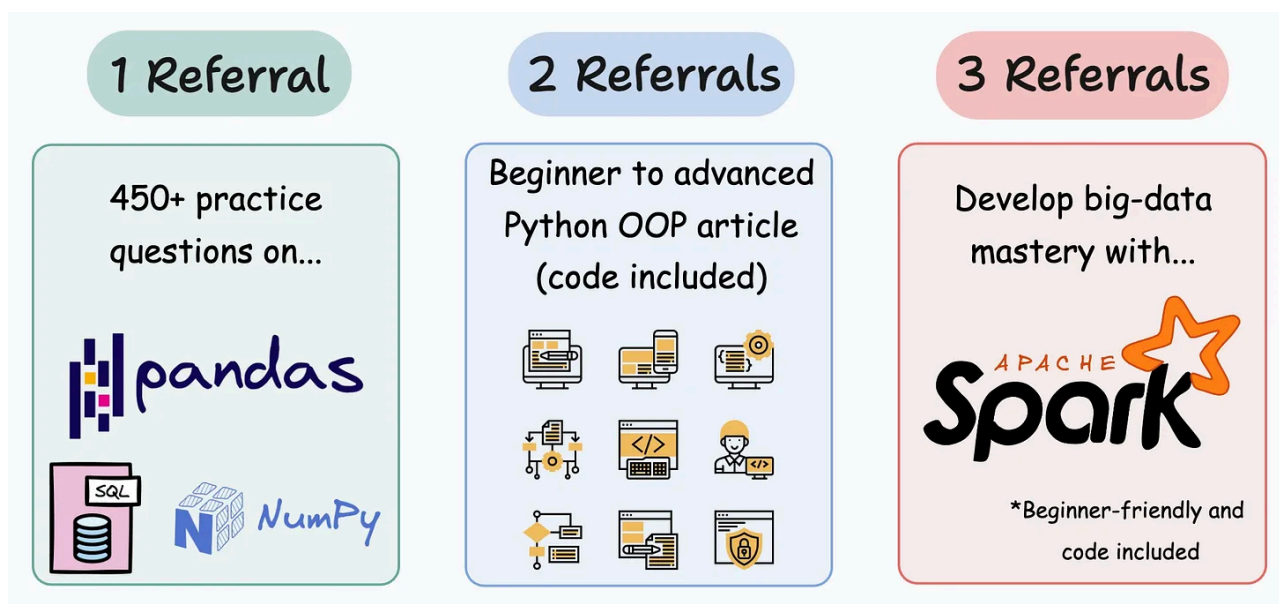
What's more, one thing I particularly dislike about inplace operations is that they inhibit method chaining as depicted below:



As a result, I never prefer using inplace operations in Pandas.

👉 Over to you: Despite this, are there still any situations where you prefer using inplace operations in Pandas?

Thanks for reading Daily Dose of Data Science!
Subscribe for free to learn something new and insightful about Python and Data Science every day. Also, get a Free Data Science PDF (550+ pages) with 320+ tips.

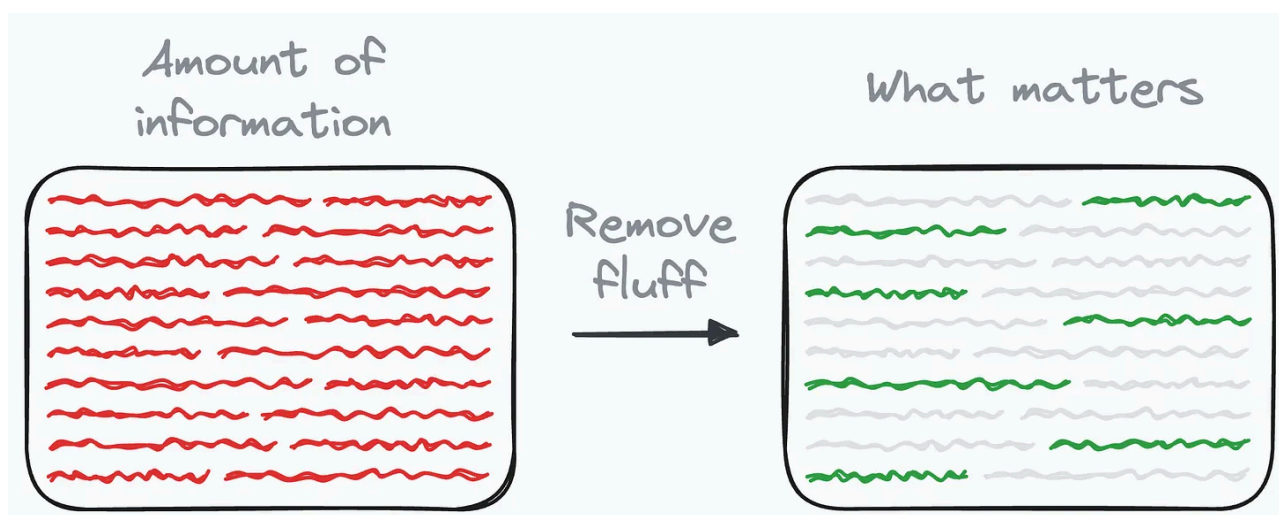


- **1 Referral:** Unlock 450+ practice questions on NumPy, Pandas, and SQL.
- **2 Referrals:** Get access to advanced Python OOP deep dive.
- **3 Referrals:** Get access to the PySpark deep dive for big-data mastery.

Get your unique referral link:

Are you overwhelmed with the amount of information in ML/DS?

Every week, I publish no-fluff deep dives on topics that truly matter to your skills for ML/DS roles.



For instance:

- [A Beginner-friendly Introduction to Kolmogorov Arnold Networks \(KANs\).](#)
- [Implementing KANs From Scratch Using PyTorch.](#)
- [A Practical Guide to Scaling ML Model Training](#)
- [5 Must-Know Ways to Test ML Models in Production \(Implementation Included\).](#)
- [A Beginner-Friendly Guide to Multi-GPU Model Training.](#)
- [Understanding LoRA-derived Techniques for Optimal LLM Fine-tuning](#)
- [8 Fatal \(Yet Non-obvious\) Pitfalls and Cautionary Measures in Data Science](#)
- [Implementing Parallelized CUDA Programs From Scratch Using CUDA Programming](#)
- [You Are Probably Building Inconsistent Classification Models Without Even Realizing.](#)
- And many many more.

Join below to unlock all full articles:

SPONSOR US

Get your product in front of 79,000 data scientists and other tech professionals.

Our newsletter puts your products and services directly in front of an audience that matters — thousands of leaders, senior data scientists, machine learning engineers, data analysts, etc., who have influence over significant tech decisions and big purchases.

To ensure your product reaches this influential audience, reserve your space [here](#) or reply to this email to ensure your product reaches this influential audience.



29 Likes

[← Previous](#)[Next →](#)

1 Comment



Mihály Nemes Jun 19 [♥ Liked by Avi Chawla](#)

And tracemalloc is an excellent tool to prove that inplace=True does NOT even spare on memory consumption although this should be the very cause of its existence. tracemalloc shows the increase in memory usage and the maximum transient usage as well. I created a test Series:

```
import pandas as pd  
data = [ 10, 8, 10, 20, 10, 8, 9, 11, 8, 6, 11, 6]  
idx = ['b', 'a', 'b', 'c', 'b', 'a', 'd', 'e', 'a', 'f', 'e', 'f']  
sr = pd.Series(data, index=idx)
```

Then I sorted it with inplace set to False and True. First measurement:




```
tm.start()  
sr_1 = sr.sort_values(inplace=False)  
del sr # just to be fair, though it played no role  
print(tm.get_traced_memory()) # (6352, 10093)  
tm.stop()
```

Second measurement:

```
tm.start()  
sr.sort_values(inplace=True)  
print(tm.get_traced_memory()) # (5648, 10093)  
tm.stop()
```

I find tracemalloc a very good instrument because it shows the real increase of Python's memory consumption. When we run `memory_usage(deep=True)` on the above Series it

shows only 792 bytes. It is only the tip of the iceberg above sea level.

 LIKE (1)  REPLY  SHARE

...

© 2024 Avi Chawla · [Privacy](#) · [Terms](#) · [Collection notice](#)
[Substack](#) is the home for great culture