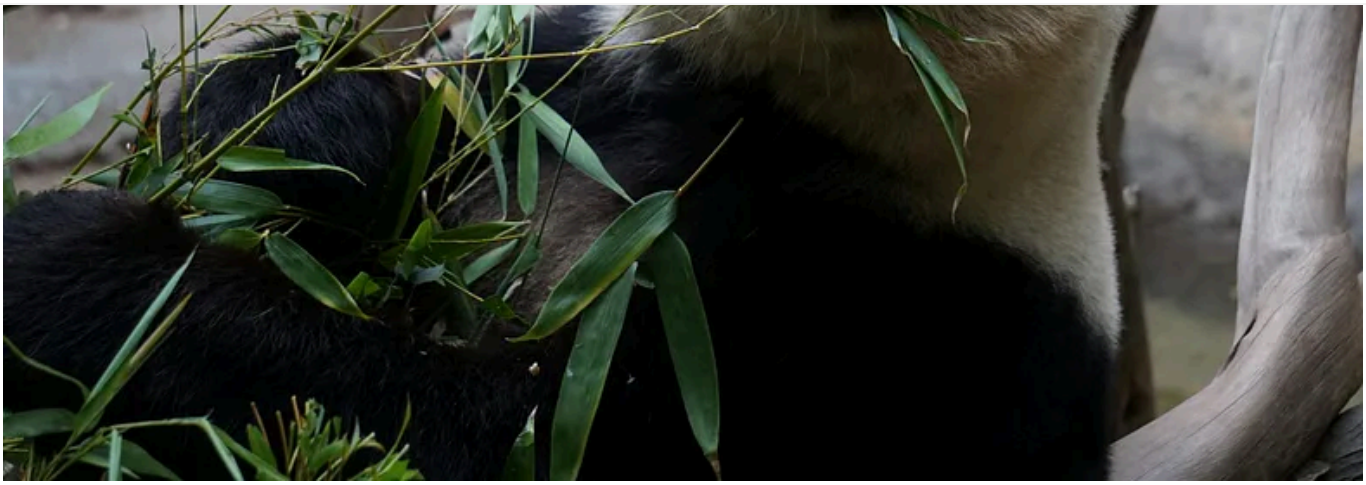
Member-only story

7 Pandas DataFrame Tricks I Wish I Knew in My Last Job

Brent Fischer · Follow

Published in Python in Plain English · 4 min read · Jan 2, 2025



Pandas for Experts

Pandas is one of the most popular Python libraries for data manipulation and analysis, but it can be overwhelming when you're starting out or under time pressure at work. Learning these tricks would have saved me a lot of frustration and time in my last job, and I'm here to share them with you so you can work smarter, not harder.

1. Use `.at` and `.iat` for Faster Access to Single Values

Accessing or modifying individual values in a DataFrame is something you'll do often. While `.loc` and `.iloc` are the most common methods, they aren't always the fastest. For single-value access, `.at` (label-based) and `.iat` (integer position-based) are optimized for speed. This can make a big difference in large datasets.

```
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Accessing a value
value = df.at[0, 'A']  # Faster than df.loc[0, 'A']

# Modifying a value
df.iat[1, 1] = 10      # Faster than df.iloc[1, 1] = 10
```

In simple terms: Use `.at` when you know the column and row labels, and `.iat` when you're working with index positions.

2. Use `query` for Cleaner Filtering

Filtering rows with conditions is a daily task. While traditional boolean indexing works, it can get messy, especially with multiple conditions. Enter the `query` method, which lets you filter rows using SQL-like syntax for readability.

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Instead of this
filtered = df[(df['A'] > 1) & (df['B'] < 6)]

# Use query for cleaner code
filtered = df.query('A > 1 & B < 6')
```

When your filtering logic gets more complex, `query` makes your code much easier to read and maintain.

3. Vectorized Operations Instead of Loops

Loops feel natural to use, but they're inefficient when working with Pandas. The library is built for vectorized operations, which are faster because they're executed in C under the hood. If you're using `for` loops or `apply` for simple column-wise calculations, you're doing it the hard way.

```
# Slow loop
df['C'] = [a + b for a, b in zip(df['A'], df['B'])]

# Fast vectorized operation
df['C'] = df['A'] + df['B']
```

Not only is the vectorized approach faster, but it's also more readable. Think of your DataFrame as a single entity rather than individual rows and columns.

4. Use `.assign` to Chain Transformations

Data cleaning often involves multiple transformations. Instead of performing one operation, assigning it back to the DataFrame, and repeating, you can use `.assign` to chain them together. This makes your code more elegant and less error-prone.

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Add or modify columns in one step
df = df.assign(
    D=df['A'] * 2,
    E=lambda x: x['B'] + 3
)
```

Now you can add or modify multiple columns without interrupting the workflow. This is especially useful when chaining multiple operations together.

5. Replace `apply` with `.map` or `.applymap` for Simpler Tasks

If you're transforming a single column, use `.map` instead of `apply`. It's faster and more concise. For element-wise transformations across an entire DataFrame, use `.applymap`. Reserve `apply` for row-wise or column-wise operations.

```
# Use map for single columns
df['A'] = df['A'].map(lambda x: x * 2)

# Use applymap for element-wise operations across the DataFrame
df = df.applymap(lambda x: x * 2)
```

This keeps your code focused and avoids unnecessary complexity.

6. Convert DataFrames to Dictionaries Efficiently

When working with APIs or exporting data, you'll often need to convert a DataFrame to a dictionary. Instead of writing custom code, use Pandas' built-in `to_dict` method. Choosing the correct `orient` argument can save you a lot of headaches.

```
# Convert rows to a list of dictionaries
data_dict = df.to_dict(orient='records')
```

This method is perfect for JSON-style outputs, where each dictionary represents a row. It's a lifesaver when integrating with external systems.

7. Use `.groupby` with Custom Aggregations

The `.groupby` function is a powerhouse for summarizing data, but did you know you can combine it with `.agg` to perform multiple custom aggregations at once? This makes it easy to produce complex summaries in just a few lines.

```
df = pd.DataFrame({'A': ['foo', 'foo', 'bar'], 'B': [1, 2, 3], 'C': [4, 5, 6]})

# Group by column 'A' and compute custom aggregations
result = df.groupby('A').agg(
    sum_B=('B', 'sum'),     # Sum of column B
    mean_C=('C', 'mean')    # Mean of column C
)
```

Instead of chaining multiple `.groupby` and aggregation calls, this allows you to calculate everything at once, making your code concise and efficient.

Thank you for being a part of the community

Before you go:

- Be sure to **clap** and **follow** the writer 👏
- Follow us: X | LinkedIn | YouTube | Newsletter | Podcast
- Check out CoFeed, the smart way to stay up-to-date with the latest in tech 🧪
- Start your own free AI-powered blog on Differ 🚀

- Join our content creators community on Discord 🧑💻
- For more content, visit plainenglish.io + stackademic.com

Pandas | Python | Dataframes | Tricks

More from Brent Fischer and Python in Plain English

In Level Up Coding by Brent Fischer

Linux for Pythonistas: Advanced File Monitoring and Automation...

In Linux, inotify (inotify stands for "inode notify") is a powerful kernel subsystem that...

Dec 12, 2024 95

In Python in Plain English by Kiran Maan

Can Mojo Really Replace Python? Let's Test It

A few days ago, I got to know about Mojo on Medium. I just made a pause and read: "Mojo...

Dec 19, 2024 697 14

In Python in Plain English by Afsalkh

Why Does round(6.5) Return 6 While round(7.5) Returns 8 in...

Pythonistas, Can You Explain This Rounding Oddity?

Dec 3, 2024 697 13

In Python in Plain English by Brent Fischer

Python Advanced: The Beginner's Guide to Ruff

As Python projects grow in size and complexity, maintaining clean and consisten...

Dec 9, 2024 20

See all from Brent Fischer

See all from Python in Plain English

Recommended from Medium

In Dev Genius by Aleksei Aleinikov

10 Ways to Work with Large Files in Python: Effortlessly Handle...

Handling large text files in Python can feel overwhelming. When files grow into...

Dec 1, 2024 304 3

De Os

Pandas Read Excel 18 Times Faster

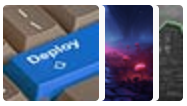
Yesterday, I had to process around 750 Excel files with approximately 165,000 rows each....

Jul 11, 2024 164 1

Lists

Coding & Development

11 stories · 964 saves



Predictive Modeling w/ Python

20 stories · 1765 saves

Practical Guides to Machine Learning

10 stories · 2138 saves

ChatGPT

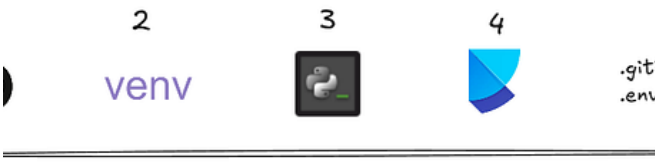
21 stories · 938 saves

Varun Singh

Python 3.14 Released — Top 5 Features You Must Know

Faster Annotations & Mind-Blowing Updates You NEED to Know!

Dec 31, 2024 62



Lorenzo Uriel

Starting a Python Project

This guide documents the steps I take to set up the foundation of any Python project, wit...

Nov 19, 2024 200 1

In Towards Data Engineering by Sandun Lakshan

10 Best Python Text User Interface (TUI) Libraries for 2025

Text-based user interfaces (TUIs) are a simple way to create interactive applications that ru...

Dec 10, 2024 36 1

In Towards Data Science by Vladimir Zhyvov

From Default Python Line Chart to Journal-Quality Infographics

Transform boring default Matplotlib line charts into stunning, customized...

Dec 30, 2024 1.1K 18

See more recommendations