9 Pandas Tricks Every Data Scientist Should Know

Uğurcan Demir · Follow

3 min read · 4 days ago





Essential Tricks for Data Manipulation That Every Geek Should Know

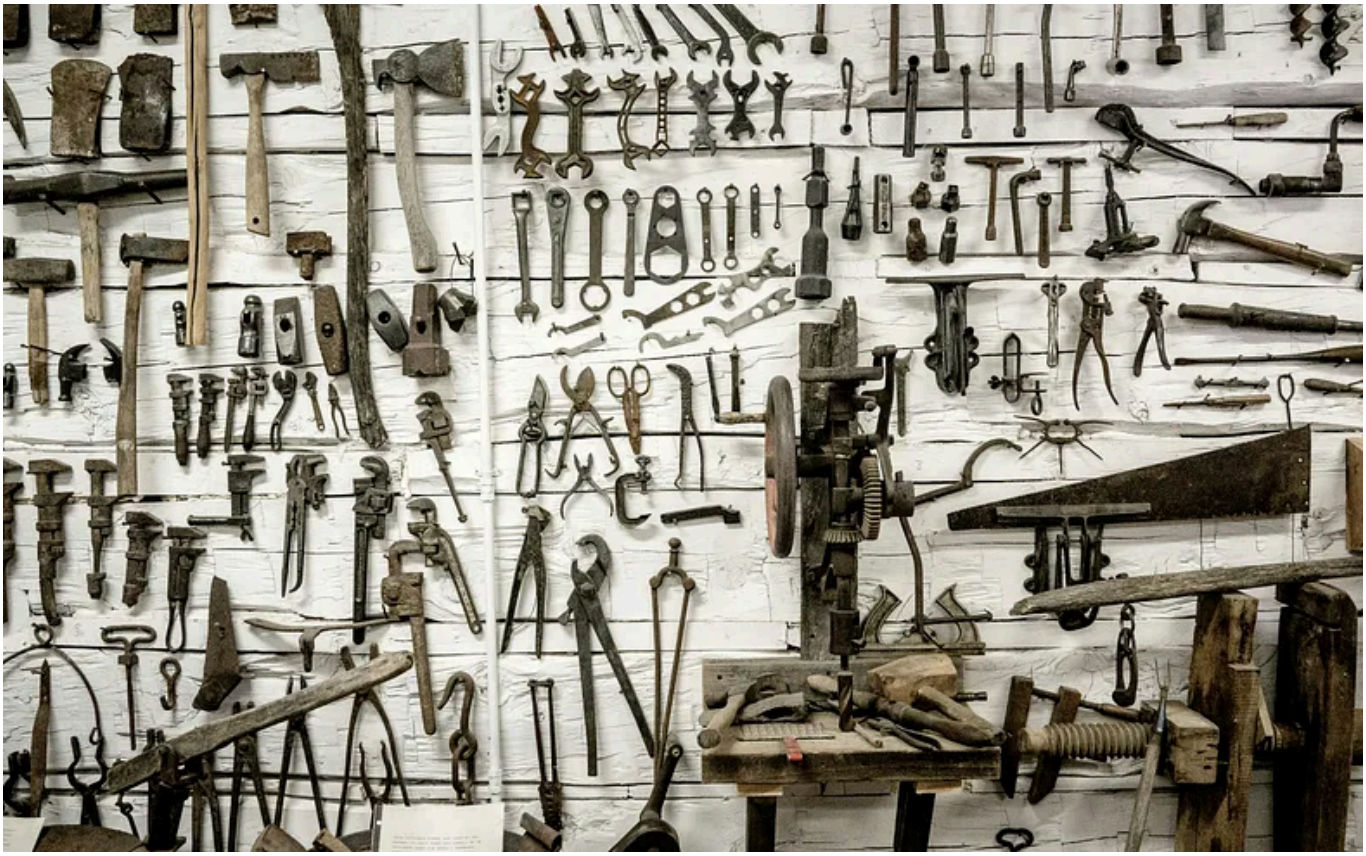


Photo by Lachlan Donald on Unsplash

Working with data can often feel like a never-ending battle. Missing values, inconsistent formatting, and sprawling datasets can make even the simplest tasks frustratingly complex. Pandas, while incredibly powerful, isn't always intuitive — long chains of code and repeated trial-and-error can leave you tearing your hair out. Without the right tools and tricks, you might find yourself drowning in inefficiencies and struggling to make sense of your data.

Mastering pandas isn't just about knowing the basics; it's about uncovering the clever shortcuts and hidden gems that make your work faster, cleaner, and more effective. A few well-placed tricks can save hours of effort, reduce bugs, and transform how you approach data analysis. These little-known features not only boost productivity but also allow you to focus on what really matters — extracting insights and making impactful decisions.

To save you from the common pitfalls and unlock the full potential of pandas, I've compiled a list of 9 powerful tricks. These aren't just neat features — they're game-changers that will simplify your workflow, improve your code, and make you fall in love with pandas all over again. Let's dive in and transform the way you handle data!

```python
import pandas as pd
import numpy as np

# Create a dummy dataset
data = {
    'region': ['East', 'West', 'East', 'West', 'East', 'West'],
    'month': ['Jan', 'Jan', 'Feb', 'Feb', 'Mar', 'Mar'],
    'sales': [200, 300, 250, 400, 300, 500],
    'price': [100, 150, 200, 250, 300, 350],
    'quantity': [2, 3, 4, 5, 6, 7],
    'email': [
        'john.doe@example.com', 'jane.doe@example.com',
        'mark.twain@example.com', 'lucy.gray@example.com',
        'harry.potter@example.com', 'hermione.granger@example.com'
    ],
    'category': ['A', 'B', 'A', 'B', 'A', 'B']
}

df = pd.DataFrame(data)
```

starting.py hosted with ❤ by GitHub view raw

1. Speed Up Complex Calculations with .eval()

The *.eval()* method simplifies calculations involving multiple columns. It avoids creating intermediate DataFrames, saving memory and improving performance, especially on large datasets.formance, especially on large datasets.

```python
# Calculate total price and profit margin
df.eval("total_price = price * quantity", inplace=True)
df.eval("profit_margin = total_price * 0.2", inplace=True)
```

eval.py hosted with ❤ by GitHub view raw

Search this file…

price	quantity	total_price	profit_margin
100	2	200	40.0
150	3	450	90.0
200	4	800	160.0
250	5	1250	250.0
300	6	1800	360.0
350	7	2450	490.0

eval_output.csv hosted with ❤ by GitHub view raw

2. Multi-Level Aggregation with .pivot_table()

.pivot_table() is a powerful tool for summarizing data. It allows you to group data by rows and columns and apply multiple aggregation functions, like sum and mean, in a single operation.

```
# Create pivot table
pivot = df.pivot_table(
    values='sales',
    index='region',
    columns='month',
    aggfunc={'sales': ['sum', 'mean']}
)
```

pivot.py hosted with ❤ by GitHub view raw

Search this file…

	mean	mean	mean	sum	sum	sum
month	Feb	Jan	Mar	Feb	Jan	Mar
region						
east	250.0	200.0	300.0	250	200	300
west	400.0	300.0	500.0	400	300	500

pivot_output.csv hosted with ❤ by GitHub view raw

3. Extract Patterns Using .str.extract()

.str.extract() is used to extract patterns from text columns using regular expressions. It's particularly useful for splitting structured text into multiple columns, like extracting usernames and domains from email addresses.

```python
# Extract username and domain
df[['username', 'domain']] = df['email'].str.extract(r'([^@]+)@([^@]+)')
```

str_ext.py hosted with ❤ by GitHub view raw

Search this file…

email	username	domain
john.doe@example.com	john.doe	example.com
jane.doe@example.com	jane.doe	example.com
mark.twain@example.com	mark.twain	example.com
lucy.gray@example.com	lucy.gray	example.com
harry.potter@example.com	harry.potter	example.com
hermione.granger@example.com	hermione.granger	example.com

str_ext_output.csv hosted with ❤ by GitHub view raw

4. Perform Flexible Multi-Column Summarization with .agg()

The *.agg()* method allows you to apply multiple aggregation functions to different columns simultaneously. It's perfect for generating concise statistical summaries of your data.

```python
# Aggregate multiple functions
summary = df.agg({
    'price': ['mean', 'sum'],
    'category': ['nunique', 'count']
})
```

agg.py hosted with ❤ by GitHub view raw

Search this file…

	price	category
mean	225.0	
sum	1350.0	
nunique		2.0
count		6.0

agg_output.csv hosted with ❤ by GitHub view raw

5. Group and Aggregate with Custom Logic Using .aggregate()

.aggregate() extends the power of .groupby() by enabling custom aggregation for grouped data. It allows you to compute tailored summaries, such as total sales and unique customers per region.

```
# Group and aggregate
grouped = df.groupby('region').aggregate(
    total_sales=('sales', 'sum'),
    avg_sales=('sales', 'mean'),
    unique_customers=('email', 'nunique')
)
```

aggregate.py hosted with ❤ by GitHub view raw

Search this file…

region	total_sales	avg_sales	unique_customers
east	750	250.0	3
west	1200	400.0	3

aggregate_output.csv hosted with ❤ by GitHub view raw

6. Apply Custom Transformations with .apply()

.apply() is highly versatile and allows you to apply custom logic to rows or columns. It's ideal for creating new columns or modifying existing ones based on conditional logic.

```python
# Apply discount logic
df['discounted_price'] = df['price'].apply(lambda x: x * 0.9 if x > 150 else x)
```

apply.py hosted with ❤ by GitHub view raw

Search this file…

price	discounted_price
100	100.0
150	150.0
200	180.0
250	225.0
300	270.0
350	315.0

apply_output.csv hosted with ❤ by GitHub view raw

7. Apply Element-Wise Transformations with .applymap()

.applymap() is used for element-wise transformations on an entire DataFrame. It's useful for applying a consistent operation, like cleaning text or standardizing data, across all cells.

```python
# Convert all string elements to lowercase
df[['region', 'category']] = df[['region', 'category']].applymap(
    lambda x: x.lower() if isinstance(x, str) else x
)
```

applymap.py hosted with ❤ by GitHub view raw

Search this file…

region	category
east	a
west	b
east	a
west	b
east	a
west	b

applymap_output.csv hosted with ❤ by GitHub view raw

8. Reshape Wide Data to Long Format with .melt()

.melt() is used to convert wide-format data into long-format, which is often required for analysis or visualization. It's especially useful for handling data with repeated measurements.

```python
# Melt data
df_melted = pd.melt(
    df,
    id_vars=['region', 'email'],
    value_vars=['price', 'quantity', 'sales'],
    var_name='attribute',
    value_name='value'
)
```

melt.py hosted with ❤ by GitHub view raw

Search this file…

region	email	attribute	value
east	john.doe@example.com	price	100
west	jane.doe@example.com	price	150
east	mark.twain@example.com	price	200
west	lucy.gray@example.com	price	250
east	harry.potter@example.com	price	300
west	hermione.granger@example.com	price	350
east	john.doe@example.com	quantity	2
west	jane.doe@example.com	quantity	3
east	mark.twain@example.com	quantity	4
west	lucy.gray@example.com	quantity	5
east	harry.potter@example.com	quantity	6
west	hermione.granger@example.com	quantity	7
east	john.doe@example.com	sales	200
west	jane.doe@example.com	sales	300
east	mark.twain@example.com	sales	250
west	lucy.gray@example.com	sales	400
east	harry.potter@example.com	sales	300
west	hermione.granger@example.com	sales	500

melt_output.csv hosted with ❤ by GitHub view raw

9. Transform Wide Data to Long Format Using .wide_to_long()

.wide_to_long() works best with wide-format data where column names are structured systematically (e.g., with a prefix like sales_). Let's restructure the original dummy dataset to make it compatible with this method.

```python
# Restructure data to fit wide-to-long format
df_wide = df.pivot(index='region', columns='month', values='sales').reset_index()
df_wide.columns = ['region', 'sales_Jan', 'sales_Feb', 'sales_Mar']

# Reshape data using wide_to_long
df_long = pd.wide_to_long(
    df_wide,
    stubnames='sales',
    i='region',
    j='month',
    sep='_',
    suffix='\\w+'
).reset_index()
```

wide_to_long.py hosted with ❤ by GitHub view raw

Search this file…

region	sales_Jan	sales_Feb	sales_Mar
east	250	200	300
west	400	300	500

wide_output.csv hosted with ❤ by GitHub view raw

Search this file…

region	month	sales
east	Jan	250
west	Jan	400
east	Feb	200
west	Feb	300
east	Mar	300
west	Mar	500

long_output.csv hosted with ❤ by GitHub view raw

In this article, I presented 9 powerful pandas tricks that can save you time and make your data manipulation workflows more efficient. From simplifying complex calculations with *.eval()* to reshaping data effortlessly with *.wide_to_long()*, these techniques are designed to help you unlock the full potential of pandas. Working with data can be time-consuming and sometimes overwhelming, but knowing these tricks can significantly streamline your tasks and improve productivity. I hope these examples inspire you to explore pandas further and integrate these techniques into your daily work.

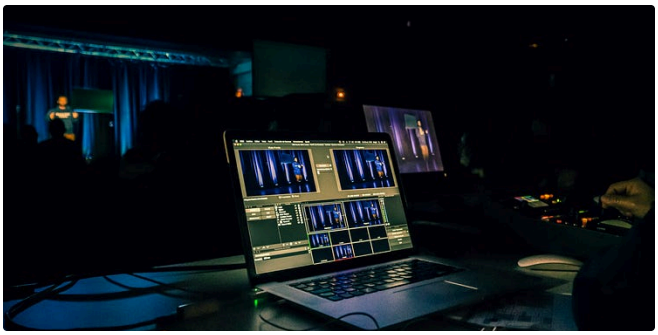
Pandas | Python | Data Science | Data Analysis | Data Wrangling

Written by Uğurcan Demir

30 Followers · 27 Following

Statistician/Econometrician. Software enthusiast.

More from Uğurcan Demir



In Better Programming by Uğurcan Demir

Creating APIs in R With Plumber

Put your R code in production

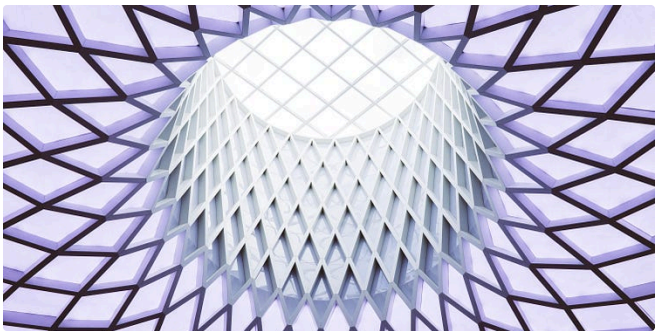
Sep 7, 2022 219 1

In Better Programming by Uğurcan Demir

R and C++ Together: An Introduction to Rcpp

Achieving high performance in R

Aug 22, 2022 198



In Better Programming by Uğurcan Demir

Learn Object-Oriented Programming Through R

An introduction to the OO system

Apr 5, 2022 111

In Better Programming by Uğurcan Demir

Handling Spatial Data in R

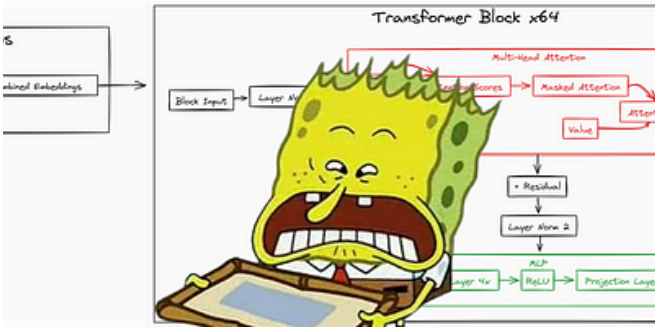
Foundations of spatial data in R

Apr 26, 2022 106

See all from Uğurcan Demir

Recommended from Medium

Austin Starks



In Level Up Coding by Fareed Khan

You are an absolute moron for believing in the hype of “AI...

All of my articles are 100% free to read. Non-members can read for free by clicking my...

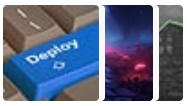
Jan 11 3.5K 158

Building a 2 Billion Parameter LLM from Scratch Using Python

It starts making sense

5d ago 752 9

Lists



Predictive Modeling w/ Python

20 stories · 1789 saves

Coding & Development

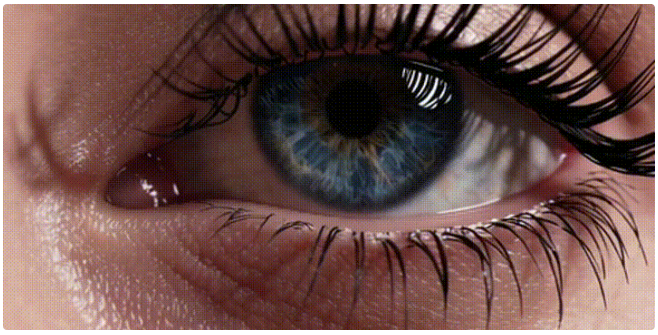
11 stories · 981 saves

Practical Guides to Machine Learning

10 stories · 2165 saves

ChatGPT prompts

51 stories · 2487 saves

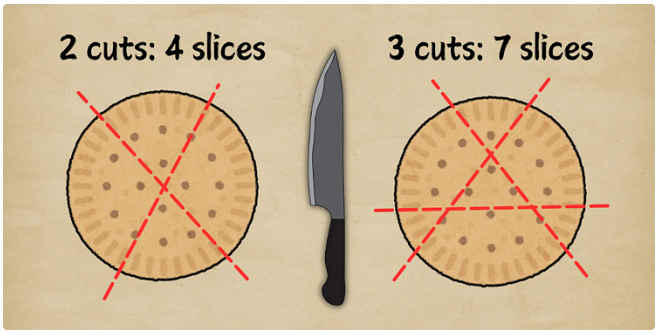


In Towards Data Science by Daniel Warfield

A Practical Exploration of Sora—Intuitively and Exhaustively...

A new cutting edge video generation tool, and the theory behind it

3d ago 120 3



In Puzzle Sphere by Fletcher Thompson

Solving the Lazy Caterer Problem

Mathematics at its best.

5d ago 115 6

In Inspire, Believe, Grow by Paul Walker

13 Terrible Christian Signs That Make You Say, ‘Wait, What?’

These were spotted out in the wild. Yes, really...

Jan 12 1.6K 29

Ritesh Gupta

The Complete Python Cheatsheet for 2025: Learn, Code, Create

All the Key Concepts, Tips, and Tricks to Boost Your Python Skills

Jan 12 74

See more recommendations