17 Pandas Trick I wish I knew Before(As a Data Scientist)

Mastering Python's Pandas: Unlock Seventeen Essential Tricks to Supercharge Your Data Science Journey

Gencay I. · Follow
Published in Towards AI · 8 min read · Jun 10

Image generated with LeonardoAI

Are you finding yourself drowning in an ocean of data?

Are complex datasets a riddle you can't crack?

Worry no more! This journey into the depths of Python's Pandas library — a vital tool in every data scientist's arsenal — will change your perspective.

By unlocking these 17 lesser-known Pandas tricks, you'll navigate the waves of data with greater ease and finesse.

With these Python, Pandas, and data manipulation tips at your disposal, you'll transition from feeling overwhelmed to overjoyed in your data science endeavours.

Believe it or not, once you've discovered these tricks, you'll wish they had been shared with you sooner. Let's embark on this enlightening journey today!

Content Table

Apply Operations Over a DataFrame

Use the `applymap` function to apply an operation over each element of the DataFrame.

```
import pandas as pd
import seaborn as sns
import numpy as np

df = sns.load_dataset('titanic')
df = df.select_dtypes(include=[np.number])  # Select only numeric columns

# Multiply all numeric values by 10
df = df.applymap(lambda x: x*10)
```

Here, we load the Titanic dataset and select only the numeric columns. The `applymap` function then multiplies each element in the DataFrame by 10.

In a real-world scenario, a Data Scientist might use this technique to perform operations on numerical data, like normalizing or standardizing the data for machine learning algorithms that are sensitive to the scale of the data.

Calculate Rolling Statistics

Use the `rolling` method followed by a statistical function to calculate the rolling statistics of a column.

```
df = sns.load_dataset('flights')
df.set_index('year', inplace=True)  # Set year as the index

# Calculate the 3-year rolling average of passengers
df['rolling_avg'] = df['passengers'].rolling(3).mean()
```

In this case, we load the Flights dataset and calculate a 3-year rolling average of the number of passengers. This technique is useful in time series analysis to smooth out short-term fluctuations and highlight longer-term trends or cycles.

Chain Methods Together

Chain methods together in one statement to make your code more readable and efficient.

```
df = sns.load_dataset('titanic')

df = (df
      .rename(columns={'class': 'passenger_class'})
      .assign(age_in_months=lambda x: x.age*12))
```

Here, we load the Titanic dataset, rename the `class` column to `passenger_class`, and then create a new column `age_in_months` by multiplying the `age` column by 12. All of this is done in one statement using method chaining.

This approach is common in Data Science workflows where multiple transformations need to be done on a dataset before analysis or modeling.

Dropping Columns with Missing Data

Use the `dropna` method with the `axis` parameter set to 1 to drop any column containing missing values.

```
df = sns.load_dataset('titanic')

# Drop columns with missing values
df = df.dropna(axis=1)
```

This code loads the Titanic dataset and removes any columns that contain missing values using the `dropna` method.

In real-world datasets, it is quite common to have missing data. Dropping columns with missing data is one approach to handle such situations, although it might not always be the best approach depending on the nature and amount of missing data.

Using Categories for Efficient Memory Use

If a column contains only a few different values, change its datatype to `category` to save memory.

```
df = sns.load_dataset('titanic')

# Convert 'sex' and 'embarked' to category datatype
df[['sex', 'embarked']] = df[['sex', 'embarked']].astype('category')
```

In this example, we load the Titanic dataset and convert the `sex` and `embarked` columns to category datatype. When dealing with large datasets, converting columns with a limited number of unique values to category datatype can help save memory.

This is especially useful in scenarios where the Data Scientist has to handle datasets that are close to the memory limit of the system.

Finding Most Frequent Values

Use the `value_counts` method to find the most frequent values in a column.

```
df = sns.load_dataset('titanic')

# Find most frequent values in 'embark_town'
most_frequent = df['embark_town'].value_counts().idxmax()
print(f"Most frequent embark town: {most_frequent}")
```

In this example, we load the Titanic dataset and find the most frequently occurring value in the ‘embark_town’ column using the `value_counts` method. The `idxmax` method is then used to return the most frequent value.

This technique can be useful for understanding the distribution of categorical data and may help to guide data cleaning or feature engineering processes.

Aggregation

Use the `agg` function to apply different aggregations to different columns of a DataFrame.

```
df = sns.load_dataset('titanic')

# Apply different aggregations to different columns
aggregated = df.agg({
    'age': ['min', 'max', 'mean'],
    'fare': ['sum', 'mean']
})
print(aggregated)
```

Here, we load the Titanic dataset and apply different aggregations to different columns. The `agg` function is used to apply a minimum, maximum, and mean calculation to the 'age' column, and a sum and mean calculation to the 'fare' column.

Aggregation is a common operation in Data Science, particularly when performing exploratory data analysis or preparing data for visualizations or modeling.

Sorting by Multiple Columns

Use the `sort_values` method with a list of column names to sort by multiple columns.

```
df = sns.load_dataset('titanic')

# Sort by 'class' and then 'fare'
df = df.sort_values(by=['class', 'fare'])
```

In this code snippet, we load the Titanic dataset and sort the DataFrame first by ‘class’ and then by ‘fare’ using the `sort_values` method. This method is useful when you want to order the data by multiple criteria.

Sorting by multiple columns is often useful in Data Science when preparing data for analysis or visualization or when looking for specific patterns or trends in the data.

Filtering with the `isin` Method

Use the `isin` method to filter rows based on a list of values.

```
df = sns.load_dataset('titanic')

# Filter rows where 'class' is either 'First' or 'Second'
df = df[df['class'].isin(['First', 'Second'])]
```

Here, we load the Titanic dataset and use the `isin` method to filter rows where 'class' is either 'First' or 'Second'. This method can be very handy when you want to filter data based on multiple values of a categorical variable.

This method is useful when working with categorical data, and you want to select rows that belong to certain categories. It is more concise than using multiple logical conditions with the `&` operator.

Accessing Grouped Elements

Use the `get_group` method after a `groupby` operation to access the data for a particular group.

```
df = sns.load_dataset('titanic')

# Group by 'class' and get data for 'First' class
grouped = df.groupby('class')
first_class_data = grouped.get_group('First')
```

In this example, we load the Titanic dataset, group the data by ‘class’, and then access the data for the ‘First’ class using the `get_group` method.

The `groupby` method is a powerful tool for segmenting data into groups to perform analyses. The `get_group` method allows you to access the data for a specific group, which can be useful when performing further analyses or transformations on specific groups.

Selecting Random Samples

Use the `sample` method to select a random sample of rows or columns from a DataFrame.

```
df = sns.load_dataset('titanic')

# Select a random sample of 5 rows
sample = df.sample(5)
```

This code loads the Titanic dataset and selects a random sample of 5 rows from the DataFrame. The `sample` method is used for this purpose.

Random sampling is often used in Data Science when working with large datasets to create smaller, more manageable datasets for exploratory analysis or prototyping models.

Setting Column as Index

Use the `set_index` method to set a column as the index of the DataFrame.

```
df = sns.load_dataset('titanic')

# Set 'class' as the index
df.set_index('class', inplace=True)
```

In this example, we load the Titanic dataset and set ‘class’ as the index of the DataFrame using the `set_index` method.

Setting a column as an index can be very useful in Data Science, as it allows for faster data retrieval and can make the DataFrame easier to read and manipulate, especially when dealing with time-series data or data with a unique identifier.

Renaming All Columns

Use the `rename` method with a function to rename all the columns in the DataFrame.

```
df = sns.load_dataset('titanic')

# Rename all columns to lower case
df.columns = df.columns.str.lower()
```

Here, we load the Titanic dataset and rename all columns to lowercase. This is a simple example of how you can modify column names using vectorized string operations.

Renaming columns is often necessary in Data Science, as it makes the DataFrame easier to work with and understand.

Finding Unique Values in a Column

Use the `unique` method to get an array of unique values from a column.

```
df = sns.load_dataset('titanic')

# Get unique values in 'class'
unique_classes = df['class'].unique()
```

This code loads the Titanic dataset and finds all unique values in the 'class' column. This is done using the `unique` method.

Finding unique values in a column is a common operation in Data Science as it gives you an understanding of the different categories or groups in your data.

Splitting a Column Using `str.split`

Split a string column into multiple columns.

```
df = sns.load_dataset('planets')

# Split 'method' into two columns
df[['method_1', 'method_2']] = df['method'].str.split(' ', 1, expand=True)
```

n this example, we load the Planets dataset and split the 'method' column into two new columns, 'method_1' and 'method_2'. This is done using the `str.split` method.

Splitting string columns is a common task when the data within a single column should be distributed over multiple columns for further analysis or preprocessing.

Using the `at` and `iat` for Fast Access

Use `at` and `iat` to access a scalar value, which is faster than `loc` and `iloc`.

```
df = sns.load_dataset('titanic')

# Get value at specific location
value = df.at[5, 'age']
```

Here, we load the Titanic dataset and retrieve a single value from the DataFrame using the `at` method.

The `at` and `iat` methods provide a faster way to access scalar values compared to `loc` and `iloc` and are particularly useful when you need to retrieve or set a single value from a large DataFrame.

Pivot Table

Use the `pivot_table` method to create a pivot table from the DataFrame data.

```
df = sns.load_dataset('titanic')

# Create a pivot table
pivot_table = df.pivot_table(values='survived', index='class', columns='sex')
```

In this example, we load the Titanic dataset and create a pivot table that shows the survival rate by class and sex.

Pivot tables are useful in Data Science for summarizing and grouping data, and are often used in exploratory data analysis to get a different perspective on the data.

My Cheat Sheets and Source Codes.

If you've made it this far, thank you!

I continually update and add new Cheat Sheets and Source Codes for your benefit. Recently, I crafted a ChatGPT cheat sheet, and honestly, I can't recall a day when I haven't used ChatGPT since its release.

Also, here is my E-Book, which explains, how Machine Learning can be learned by using ChatGPT.

Feel free to select one of the Cheat Sheets or projects for me to send you by completing the forms below.

Here is the ChatGPT cheat sheet.

Here is my NumPy cheat sheet.

Here is the source code of the "How to be a Billionaire" data project.

Here is the source code of the "Classification Task with 6 Different Algorithms using Python" data project.

Here is the source code of the "Decision Tree in Energy Efficiency Analysis" data project.

Here is the source code of the "DataDrivenInvestor 2022 Articles Analysis" data project.

In case you're not yet a Medium member and want to expand your knowledge through reading, here's my referral link.

Here is my E-Book: How to Learn Machine Learning with ChatGPT?

> "Machine learning is the last invention that humanity will ever need to make." Nick Bostrom

Python Pandas Data Science Data Scientist Data Manipulation

237 1

Written by Gencay I.

Follow

1.5K Followers · Writer for Towards AI

150K+ Views on Medium | AI, DS, Machine Learning Writer | BSc Engineer & MSc AI | Join 1K+ people for latest news! https://gencay.ck.page/f0a11de9ba

More from Gencay I. and Towards AI

Gencay I. in Level Up Coding

5 ChatGPT plugins That Will put you ahead of 99% of Data…

Elevate Your Data Science Game with ChatGPT Plugins

· 7 min read · Jul 7

617 9

Dr. Mandar Karhade, MD. PhD. in Towards AI

Better than GPT-4 for SQL queries: NSQL (Fully…

NSQL is a new family of open-source large foundation models (FMs) designed…

· 8 min read · Jul 7

420 6

Bex T. in Towards AI

10 Advanced Matplotlib Concepts You Must Know To Create Killer…

Become Leonardo da Matplotlib

· 9 min read · Jun 30

746 7

Gencay I. in Level Up Coding

I tried 100+ ChatGPT prompts. Here are the best 4!

Unleashing the Power of ChatGPT: Discover the Top 4 Prompts for Amazing Results

· 8 min read · Jul 12

378 4

See all from Gencay I.

See all from Towards AI

Recommended from Medium

Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt…

· 26 min read · Jun 22

4.1K 38

Anmol Tomar in CodeX

16 Python Tricks To Learn Before You Write Your Next Code

Tricks that will make your life easier as a python developer

· 5 min read · Feb 23

1.2K 11

Lists

Predictive Modeling w/ Python
18 stories · 232 saves

Coding & Development
11 stories · 95 saves

New_Reading_List
174 stories · 61 saves

Practical Guides to Machine Learning
10 stories · 242 saves

Cornellius Yudha Wijaya in Towards AI

3 Pandas Functions for DataFrame Merging

Learn how Pandas merging functions work with code examples

· 14 min read · 5 days ago

280 4

Matt Chapman in Towards Data Science

The Portfolio that Got Me a Data Scientist Job

Spoiler alert: It was surprisingly easy (and free) to make

· 10 min read · Mar 24

4.1K 72

Yang Zhou in TechToFreedom

9 Fabulous Python Tricks That Make Your Code More Elegant

Pythonic is a synonym for elegant

· 5 min read · Nov 13, 2022

2.4K 23

Gabe A, M.Sc. in Level Up Coding

🐼Introducing PandasAI: The Generative AI Python Library 🐼

Pandas AI is an additional Python library that enhances Pandas, the widely-used data…

· 11 min read · May 16

1.5K 17

See more recommendations

Help Status Writers Blog Careers Privacy Terms About Text to speech Teams