**The ultimate reference for clean Pandas code**

**A clean way to clean data**

Pandas can transform even the messiest data into pristine machine learning datasets. The process itself, though, can be quite messy.

Pandas code can be hard to read for a number of reasons. For one thing, there are many different ways of accomplishing the same basic tasks in Pandas. Subsetting data, adding new columns, dropping columns, removing null values and many other processes can be done in a number of different ways, which leads to inconsistent and messy code.

Managing the order of data cleaning steps can also be challenging in Pandas. Most of my data cleaning code earlier in my career looked like this:

# Import data  
df\_raw = pd.read\_csv("path/to/data/file.csv")  
  
# Clean data  
df\_raw["id"] = df\_raw["id"].astype(str)  
df\_merged = df\_raw.merge(df2, on="id")  
df\_final = df\_merged.drop(columns=["col\_5", "col\_6", "col\_7"])  
  
# Investigate data  
df\_agg = df\_final.groupby("id").size()

“Spaghetti code,” like the example above, is difficult to interpret and debug. Furthermore, having so many data frames in the namespace uses up memory and can be especially problematic when working with large datasets.

Finally, Pandas code can get messy because many times it’s written in a hurry. Whether you’re itching to build your model and need to quickly clean your dataset beforehand or you have a fresh batch of output data that you want to analyze, as a data scientist Pandas is generally a means to an end.

So what’s the secret to writing clean Pandas code once and for all? Two words: **method chaining**.

In this article, I share with you a curated collection of clean Pandas methods that I use to preprocess, investigate, aggregate, and analyze Twitter data that I use, in a [separate project](https://github.com/m-newhauser/distilbert-senator-tweets), to train a Transformer model. Drawing on these examples will expand your understanding of method chaining and serve as a reference guide for you to write your own clean Pandas code.

**The basics of clean Pandas**

The Pandas library comes with loads of built-in methods. Recall that in Python, [methods](https://docs.python.org/3/tutorial/classes.html#:~:text=A) are functions that belong to an object of a specific class and are tacked onto the object itself, like df.to\_csv(). Methods can also be chained, meaning that you can apply several methods to an object at one time.

new\_df = ( # Wrap everything in ()'s  
 original\_df # Name of data frame to modify  
 .query("text\_length > 140") # Subset based on text length  
 .sort\_values(by="text\_length") # Sort entire df by text length  
 .reset\_index() # Reset index of subsetted df  
)

Forcing yourself to use Pandas methods instead of operators can be frustrating at first because, for the most part, you’re relearning things you already know. But here’s why you should stick with method chaining:

* It makes code much more readable.
* It eliminates the need for multiple intermediary data frames, which saves memory.
* It’s easier to debug. Simply comment out data frame manipulations line by line to see which method is giving you problems.

**Data cleaning**

I have a data frame of raw tweets made by US Senators that I fetched via the [Twitter API v2](https://developer.twitter.com/en/docs/twitter-api) with elevated access credentials. Here’s a peek at the data:

Now let’s do some chaining to clean this up. In just this single call, we will select and drop columns, format the date column, clean the raw text of the tweets, count text length, merge two data frames, drop duplicate rows, rename columns, reorder column names, sort by date, and drop all rows where tweet length is zero.

You’re probably familiar with most of these methods. Perhaps the most important method here is .assign(), which allows you to create new columns or overwrite old ones. I primarily use the assign() method for two purposes.

1. Changing the data type of an existing column:

.assign(column\_name=original\_df["column\_name"].astype(str)

2. Applying functions to entire columns:

.assign(new\_column=original\_df["column\_name"].apply(function\_name)

Note that you can also apply multiple functions to a column sequentially.

**Data investigation**

After implementing that monster chain to the raw Twitter data, we have a tidy, readable data frame that we’d like to inspect.

The simple method .info() gives you a remarkable amount of information about your data frame, including:

* Number of rows (and the index range)
* Number of columns
* Names of columns
* Data types of columns
* Number of non-null values per column
* Memory usage

This handy one-liner to get the number of null values per column:

The .describe() method gives you an overview of the actual values and distributions of the data in each of your columns. Apply .describe() to columns based on their data type for cleaner output, as shown below:

The results from calling .describe() on dtype="object" are not especially insightful, as the columns id, username, and text contain string values rather than categorical data. However, the row values for the party column could show a potential pattern.

**Data aggregation & analysis**

Data aggregation on categorical variables is usually the first part of any analysis I perform for NLP projects. The most obvious variable to aggregate on in the tweet dataset is party.

Next, let’s move on to some more advanced aggregations. Chaining .groupby() and .agg() functions in order like this make it easier to understand the aggregation as a whole:

After applying the aggregations, the resulting index is hard to read. The .pipe() method is the clean Pandas way to apply a function to an entire data frame.

**Conclusion**

The key to writing clean Pandas code is to force yourself to use **method chaining**. Doing so ultimately makes your code more readable and interpretable, easier to debug, and even saves memory. As this article has demonstrated, you can use method chaining in every part of the data lifecycle, including cleaning, investigating, aggregating, and analyzing data. For more information on method chaining, check out the resources below.

**Resources**

* [Supporting code for this article](https://github.com/m-newhauser/clean-pandas)
* [Effective Pandas by Matt Harrison](https://www.youtube.com/watch?v=UURvPeczxJI)
* [Modern Pandas](https://tomaugspurger.github.io/modern-1-intro)
* [Pandas Documentation](https://pandas.pydata.org/docs/)

**References**

(1) M. Newhauser, [DistilBERT senator tweets](https://github.com/m-newhauser/distilbert-senator-tweets) (2022).

(2) T. Augspurger, [Modern Pandas (Part 1)](https://tomaugspurger.github.io/modern-1-intro.html) (2016).

(3) M. Harrison & T. Petrou, [Pandas 1.x cookbook: Practical recipes for scientific computing, time series analysis, and exploratory data analysis using Python (Second edition)](https://www.packtpub.com/product/pandas-1-x-cookbook-second-edition/9781839213106) (2020).

(4) Python Software Foundation, [9. Classes](https://docs.python.org/3/tutorial/classes.html#:~:text=A%20method%20is%20a%20function,%2C%20sort%2C%20and%20so%20on) (2022).

(5) Twitter, [Twitter API Documentation](https://developer.twitter.com/en/docs/twitter-api) (2022).

**Want to connect?**

Contact me on [LinkedIn](https://www.linkedin.com/in/mary-newhauser-02273551/) ✍️.