**20% of NumPy Functions that Data Scientists use 80% of the Time**

Plus a free NumPy CheatSheet from DataCamp

[[Anjolaoluwa Ajayi](https://medium.com/@dataprincess?source=post_page-----1634ec27d95e--------------------------------)](https://medium.com/@dataprincess?source=post_page-----1634ec27d95e--------------------------------)

[[GDSC Babcock Dataverse](https://medium.com/gdsc-babcock-dataverse?source=post_page-----1634ec27d95e--------------------------------)](https://medium.com/gdsc-babcock-dataverse?source=post_page-----1634ec27d95e--------------------------------)

[Anjolaoluwa Ajayi](https://medium.com/@dataprincess?source=post_page-----1634ec27d95e--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F191252a32a00&operation=register&redirect=https%3A%2F%2Fmedium.com%2Fgdsc-babcock-dataverse%2F20-of-numpy-functions-that-data-scientists-use-80-of-the-time-1634ec27d95e&user=Anjolaoluwa+Ajayi&userId=191252a32a00&source=post_page-191252a32a00----1634ec27d95e---------------------post_header-----------)

Published in

[GDSC Babcock Dataverse](https://medium.com/gdsc-babcock-dataverse?source=post_page-----1634ec27d95e--------------------------------)

·

3 min read

·

Nov 16, 2023

158

2

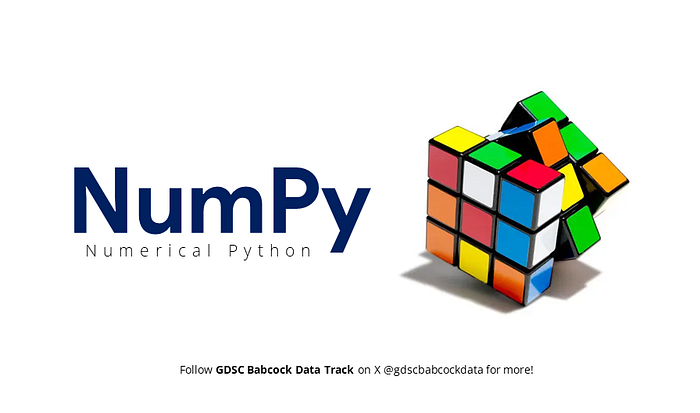


Image created by Author

As a data scientist who’s worked on over 30 projects of different sizes, scopes, and purposes… I’ll be real with you.

You may rarely use NumPy *‘directly’ for the major parts of* your workflow, instead you’d mostly be leveraging other libraries that have been built on NumPy such as Pandas, MatplotLib, and Scikit-Learn.

I know, I know what you’re thinking. But hear me out.

Despite not using NumPy functions explicitly in every project, having a solid understanding of NumPy can be a real time saver.

Many underlying operations in other libraries are implemented using NumPy so understanding how to work with arrays efficiently will make you a better data scientist cos you’ll be able to optimize your code and improve your model’s performance effortlessly.

Okay, okay, this isn’t some *‘justice for NumPy’* post; it’s more like a little list I’ve compiled that contains some NumPy functions I find myself reaching for frequently in my data science work.

But before we get to it…

**What is NumPy?**

NumPy which stands for Numerical Python is a top Python library that’s used for working with arrays (vectors and matrices — which form the very basis of the datasets you’ll be working with).

For more info about NumPy, you can visit the [NumPy Documentation](https://numpy.org/doc/stable/user/whatisnumpy.html).

While NumPy provides a whole lot of functions that you can use in your workflow, I often substitute a handful of them with Pandas and Matplotlib for more common tasks.

For instance, instead of using np.mean, np.std, etc., I find it more intuitive to employ Pandas' describe() for a comprehensive summary.

So in your own work process, you may find that you mostly need to consult NumPy functions when you need a bit more fine-tuning or when working on operations that demand a lower-level approach.

To use the NumPy library, first ensure you’ve imported it into your code like this:

import numpy as np

For the sake of this blog, assume we have loaded a dataset (of 10 rows) into our code as data that contains:

* a numerical column: numeric\_column
* another numerical column: numeric\_column1

**It’s time to bake the biscuits!**

(In Ernie’s voice)

1. np.unique(): Identifying unique values in one or more numerical columns of a dataset.

unique\_nums = np.unique(data[['numeric\_column', 'numeric\_column1']])

2. np.array(): For converting a column in a dataset into a NumPy array for numerical operations.

num\_array = np.array(data['numeric\_column'])

3. np.arange(): For generating a sequence of numbers for an index or time steps. i.e. Creating an array representing time intervals for time series analysis.

# Generate a sequence of time intervals (e.g., days, hours, etc.)  
time\_intervals = np.arange(1, 366, 7)

4. np.reshape() : For preparing data for input into machine learning models. I.e. reshaping a 1D array of pixel values into a 2D array for image processing.

# Reshape a 1D array into a 2D array (e.g., representing a 2x5 image)  
image\_2d = np.reshape(data['numeric\_column'], (2, 5))

5. np.ceil(): Rounding up elements to the nearest integer.

rounded\_up\_nums = np.ceil(data['numeric\_column'])

6. np.floor(): Rounding down elements to the nearest integer.

rounded\_down\_nums = np.floor(data['numeric\_column'])

7. np.exp(): For exponential transformation for feature engineering.

exp\_trans\_data = np.exp(data['numeric\_column'])

8. np.log(): For calculating the natural logarithm for proportional relationships. I.e. Transforming data to achieve linear relationships for regression analysis.

log\_trans\_data = np.log(data['numeric\_column'])

9. np.power(): Raising elements to a specified power. I.e. Applying power transformations to features in a machine learning model.

power\_trans\_data = np.power(data['numeric\_column'], 2)

10. np.sqrt(): For calculating square roots for scaling purposes. I.e. Scaling data using square root transformations for better model performance.

sqrt\_trans\_array = np.sqrt(data['numeric\_column'])

**And finally as promised…**

Your Free NumPy Cheatsheet from DataCamp → [link](https://www.datacamp.com/cheat-sheet/numpy-cheat-sheet-data-analysis-in-python).

**Call-to-Action**

Please leave as many claps as you please (up to 50) if you enjoyed this article and let me know in the comments what other NumPy functions you use regularly.

**I’ll be very happy if you**

Follow [GDSC Babcock DataVerse](https://medium.com/gdsc-babcock-dataverse) for more data-related articles.

*Bye for now :)*