

[Get started](#)[Open in app](#)

towards
data science

[Follow](#)

593K Followers



Write Clean Python Code Using Pipes

A Short and Clean Approach to Processing Iterables



Khuyen Tran · 3 days ago · 5 min read ★

Motivation

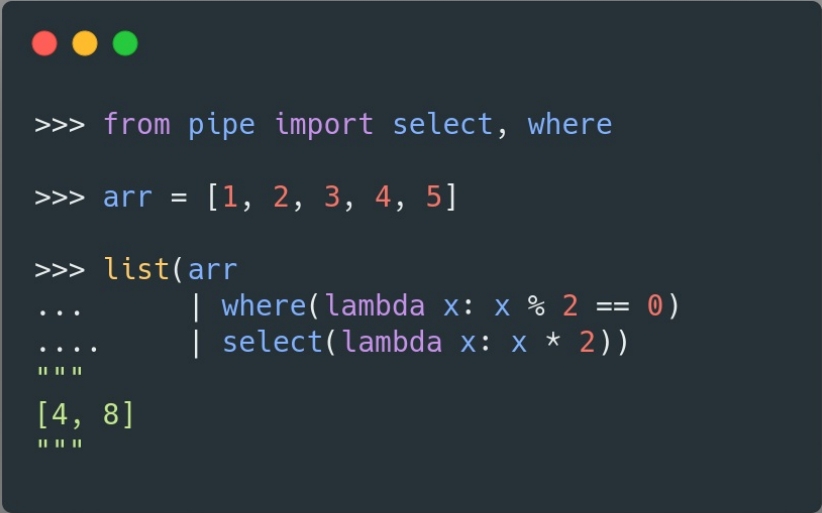
`map` and `filter` are two efficient Python methods to work with iterables. However, the code can look messy if you use both `map` and `filter` at the same time.



```
>>> arr = [1, 2, 3, 4, 5]
>>> list(map(lambda x: x * 2, filter(lambda x: x % 2 == 0, arr)))
'''
[4, 8]
'''
```

Image by Author

Wouldn't be nice if you can use pipes `|` to apply multiple methods on an iterable like below?



```
>>> from pipe import select, where

>>> arr = [1, 2, 3, 4, 5]

>>> list(arr
...     | where(lambda x: x % 2 == 0)
...     | select(lambda x: x * 2))
"""
[4, 8]
"""
```

Image by Author

The library Pipe allows you to do exactly that.

What is Pipe?

Pipe is a Python library that enables you to use pipes in Python. A pipe (`|`) passes the results of one method to another method.

I like Pipe because it makes my code look cleaner when applying multiple methods to a Python iterable. Since Pipe only provides a few methods, it is also very easy to learn Pipe. In this article, I will show you some methods I found the most useful.

To install Pipe, type:

```
pip install pipe
```

Where — Filter Elements in an Iterable

Similar to SQL, Pipe's `where` method can also be used to filter elements in an iterable.

Image by Author

Select — Apply a Function to an Iterable

The `select` method is similar to the `map` method. `select` applies a method to each element of an iterable.

In the code below, I use `select` to multiply each element in the list by 2.

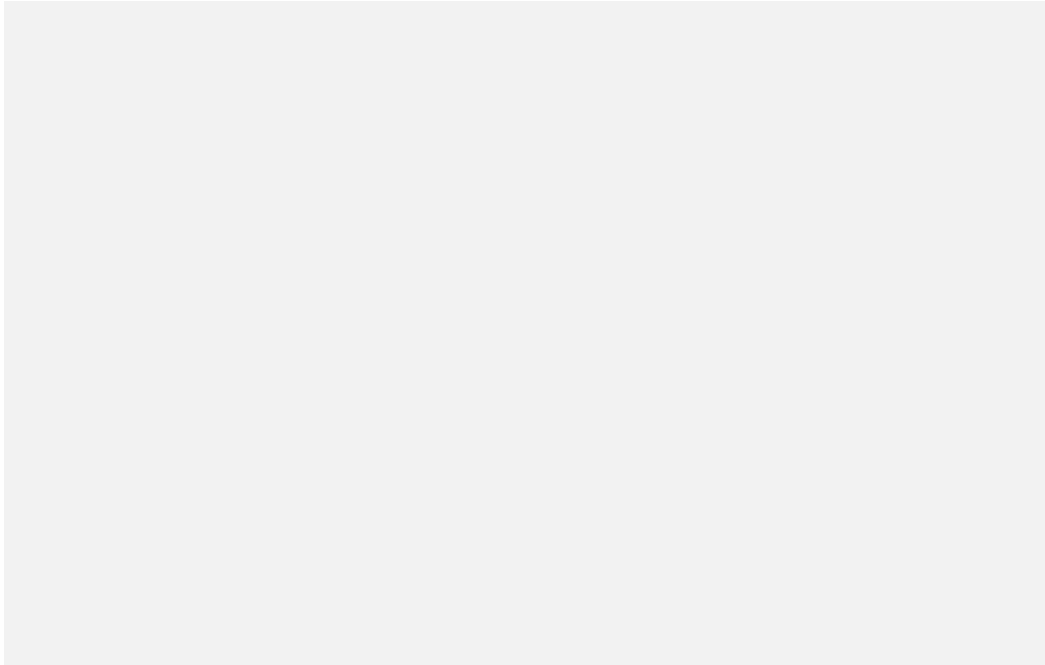
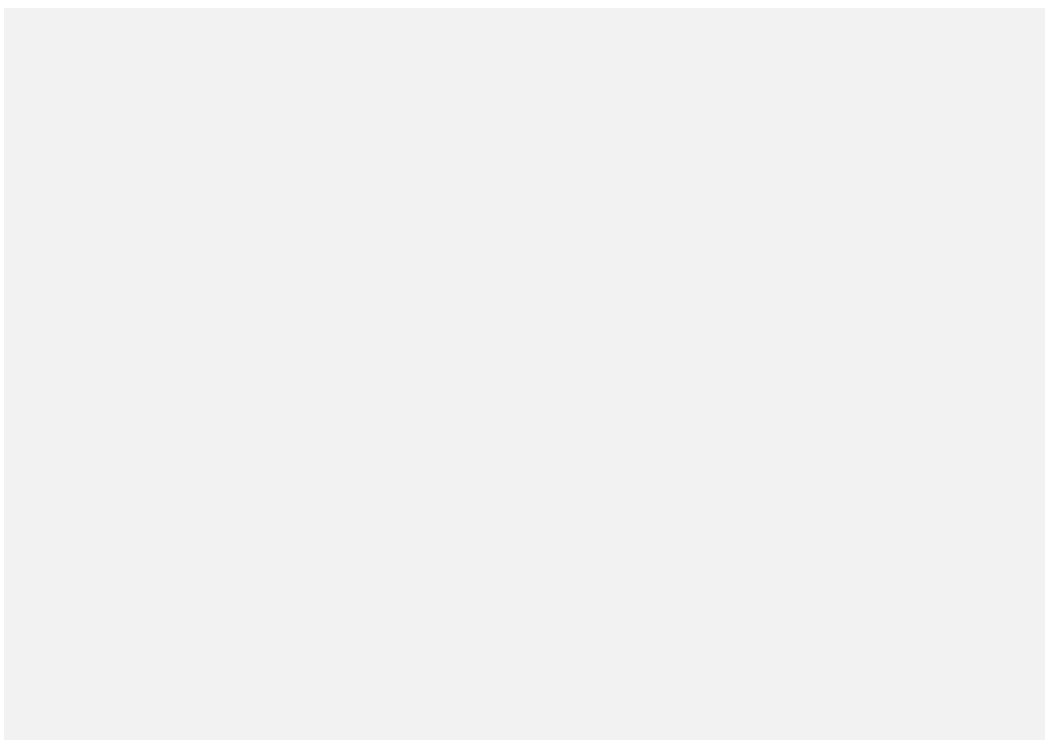


Image by Author

Now, you might wonder: Why do we need the methods `where` and `select` if they have the same functionalities as `map` and `filter` ?

It is because you can insert one method after another method using pipes. As a result, using pipes removes nested parentheses and makes the code more readable.



Unfold Iterables

chain — Chain a Sequence of Iterables

It can be a pain to work with a nested iterable. Luckily, you can use `chain` to chain a sequence of iterables.

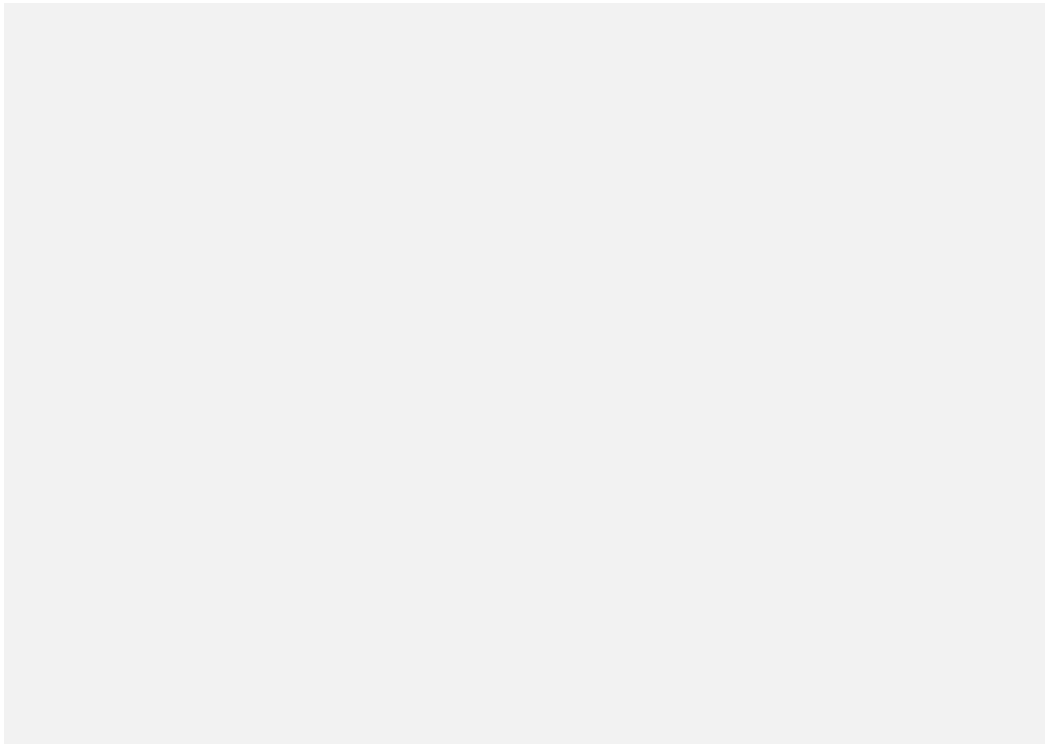
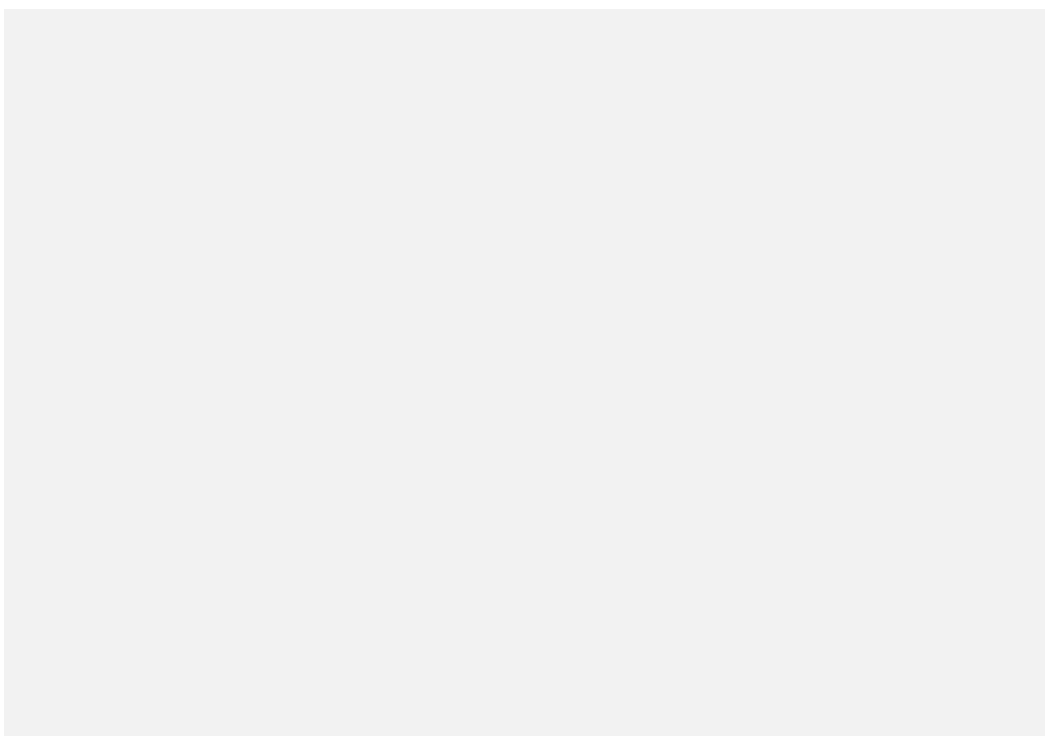


Image by Author

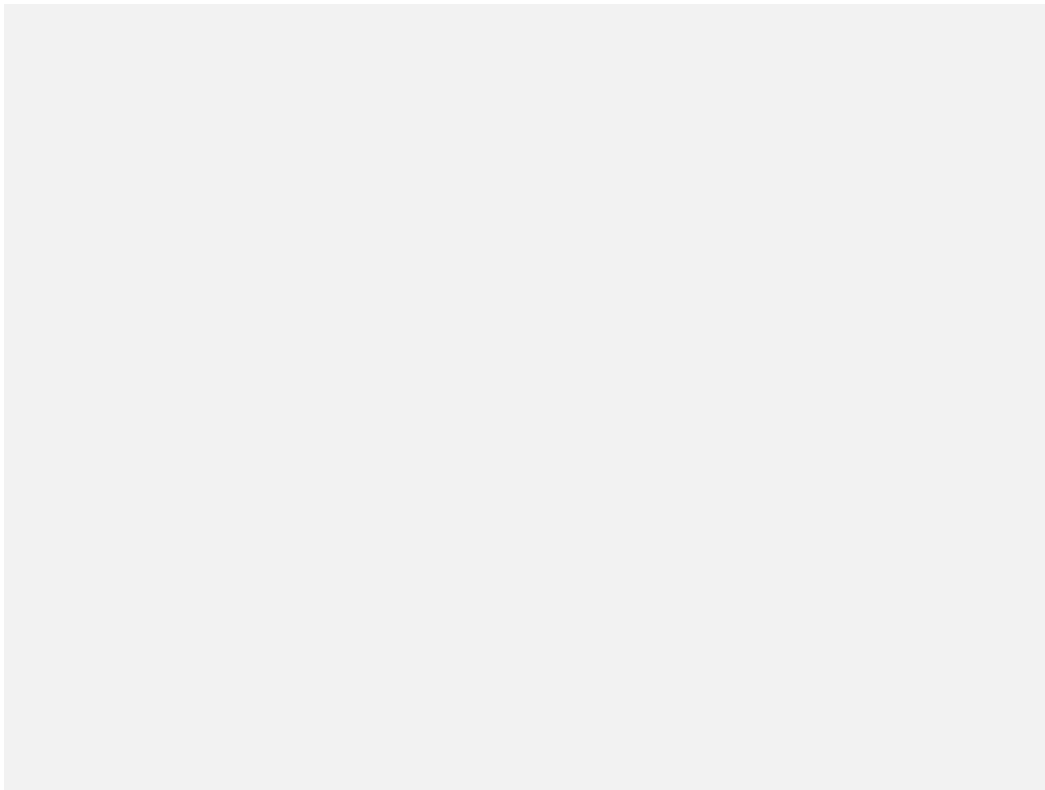
Even though the iterable is less nested after applying `chain`, we still have a nested list. To deal with a deeply nested list, we can use `traverse` instead.

traverse — Recursively Unfold Iterables

The `traverse` method can be used to recursively unfold iterables. Thus, you can use this method to turn a deeply nested list into a flat list.



Let's integrate this method with the `select` method to get the values of a dictionary and flatten the list.



Pretty cool!

Group Elements in a List

Sometimes, it might be useful to group elements in a list using a certain function. That could be easily done with the `groupby` method.

To see how this method works, let's turn a list of numbers into a dictionary that groups numbers based on whether they are even or odd.

```
>>> from pipe import groupby, select

>>> list(
...     (1, 2, 3, 4, 5, 6, 7, 8, 9)
...     | groupby(lambda x: "Even" if x % 2==0 else "Odd")
...     | select(lambda x: {x[0]: list(x[1])})
... )

"""
[{'Even': [2, 4, 6, 8]}, {'Odd': [1, 3, 5, 7, 9]}]
"""
```

Image by Author

In the code above, we use `groupby` to group numbers into the `Even` group and the `Odd` group. The output after applying this method looks like the below:

```
[('Even', <itertools._grouper at 0x7fbea8030550>),  
 ('Odd', <itertools._grouper at 0x7fbea80309a0>)]
```

Next, we use `select` to turn a list of tuples into a list of dictionaries whose keys are the first elements in the tuples and values are the second elements in the tuples.

```
[{'Even': [2, 4, 6, 8]}, {'Odd': [1, 3, 5, 7, 9]}]
```

Cool! To get only the values that are greater than 2, we can add the `where` method inside the `select` method:

```
>>> from pipe import groupby, select, where  
  
>>> list(  
...     (1, 2, 3, 4, 5, 6, 7, 8, 9)  
...     | groupby(lambda x: "Even" if x % 2 == 0 else "Odd")  
...     | select(lambda x: {x[0]: list(x[1] | where(lambda x: x > 2))})  
... )  
  
"""  
[{'Even': [4, 6, 8]}, {'Odd': [3, 5, 7, 9]}]  
"""
```

Image by Author

Note that there are no longer `2` and `1` in the outputs.

dedup — Deduplicate Values Using a Key

The `dedup` method removes duplicates in a list.

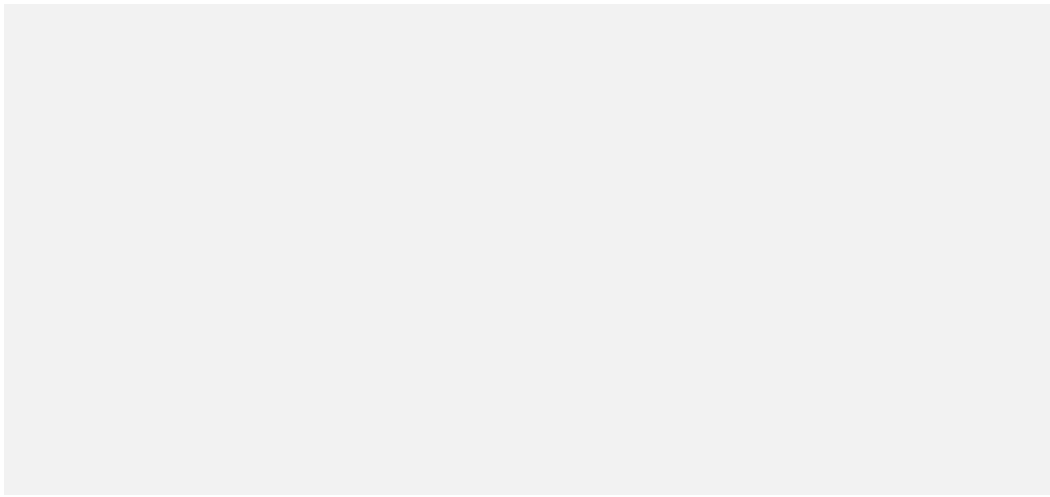


Image by Author

That might not sound interesting since the `set` method can do the same thing. However, this method is more flexible since it enables you to get unique elements using a key.

For example, you can use this method to get a unique element that is smaller than 5 and another unique element that is larger than or equal to 5.

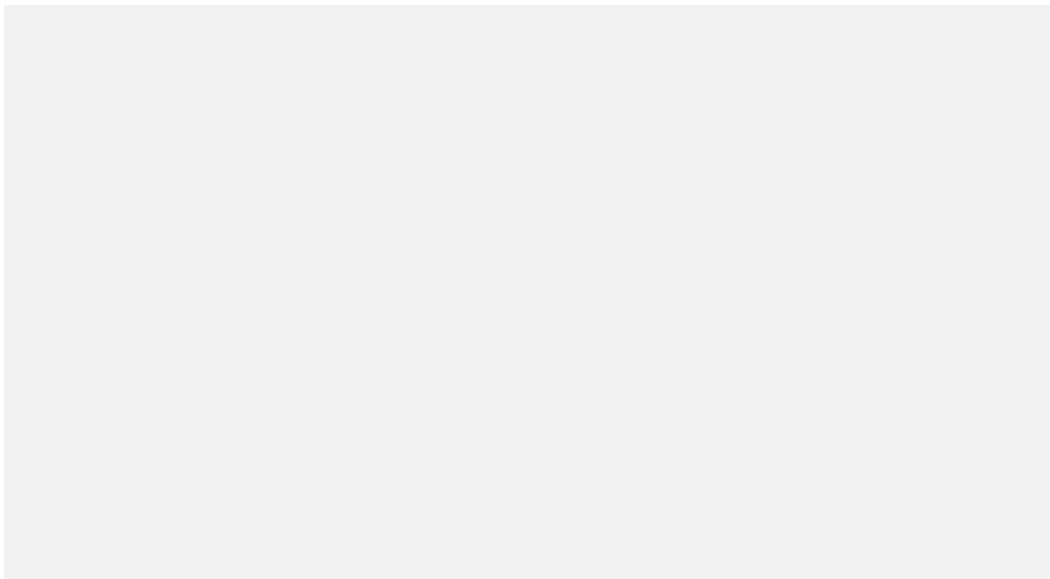
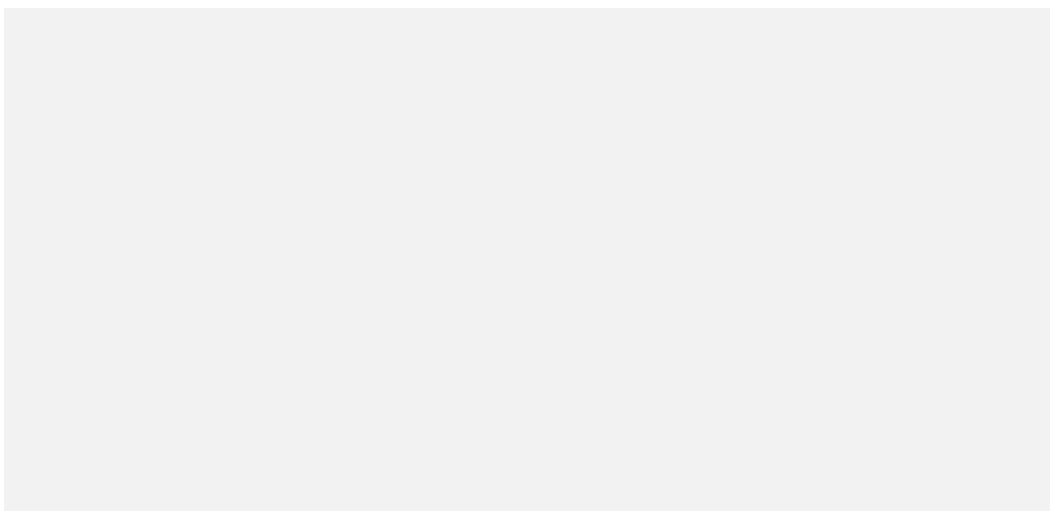


Image by Author

Now, let's combine this method with `select` and `where` to get the values of a dictionary that has duplicated keys and `None` values.



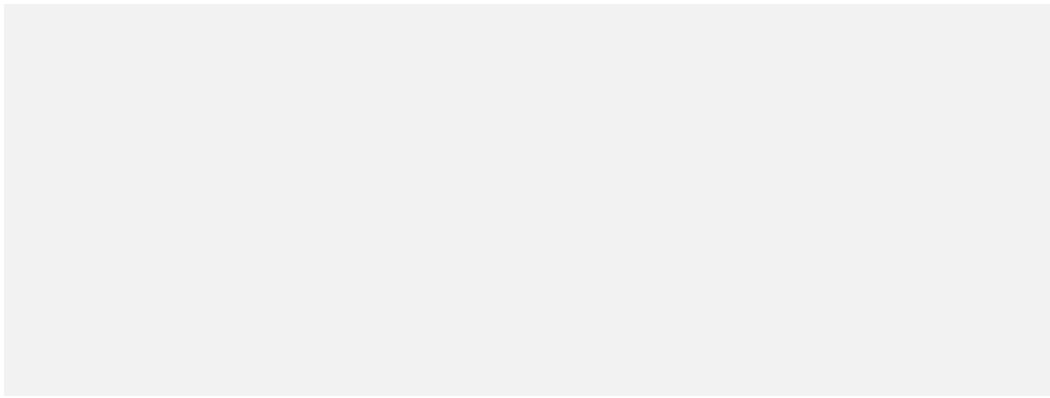


Image by Author

In the code above, we:

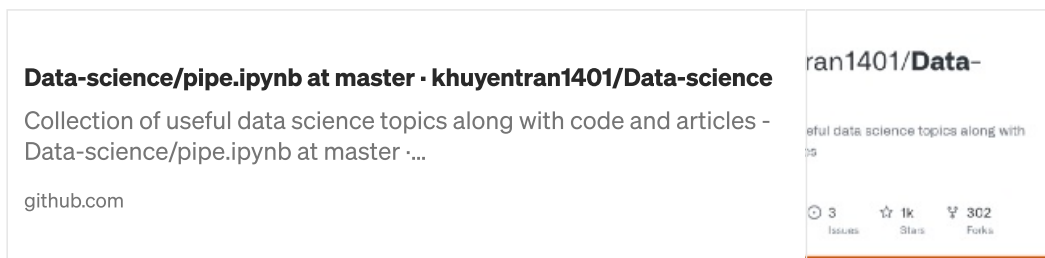
- remove items with the same `name`
- get the values of `count`
- only choose the values that are integers.

Within a few lines of code, we can apply multiple methods to an iterable while still keeping the code clean. Pretty cool, isn't it?

Conclusion

Congratulations! You have just learned how to use pipe to keep your code clean and short. I hope this article will give you the knowledge to turn complicated operations on an iterable into one simple line of code.

Feel free to play and fork the source code of this article here:

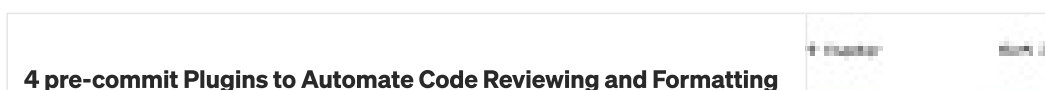
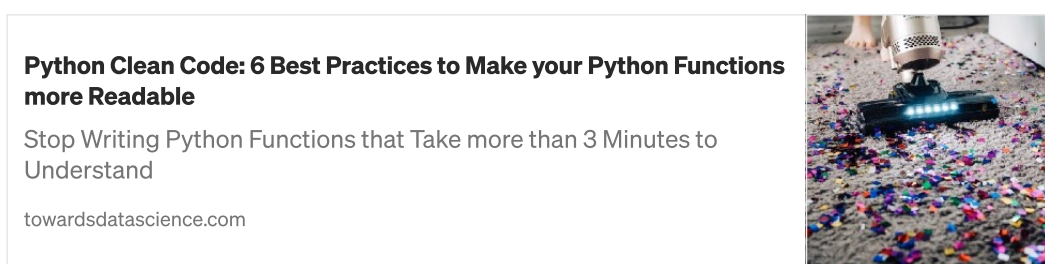


• • •

I like to write about basic data science concepts and play with different algorithms and data science tools. You could connect with me on [LinkedIn](#) and [Twitter](#).

Star [this repo](#) if you want to check out the codes for all of the articles I have written.

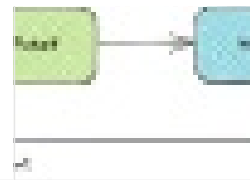
Follow me on Medium to stay informed with my latest data science articles like these:



in Python

Write High-Quality Code with black, flake8, isort, and interrogate

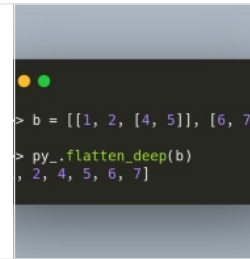
towardsdatascience.com



Pydash: A Kitchen Sink of Missing Python Utilities

Doing Python in a Functional Way

towardsdatascience.com



3 Python Tricks to Read, Create, and Run Multiple Files Automatically

Automate Boring Stuff with Python and Bash For Loop

towardsdatascience.com



Pipeline

Python

Programming

Python3

Coding