## You

Following is the simple 1 column text file. I want you to give me python script to convert it into numpy array - one element for each row below - IT will be simple one dimensional array.

Column1
Scheme Code;Scheme Name;ISIN Div Payout/ISIN Growth;ISIN Div Reinvestment;Net Asset Value;Repurchase Price;Sale Price;Date

Open Ended Schemes ( Money Market )

Taurus Mutual Fund
139619;Taurus Investor Education Pool - Unclaimed Dividend - Growth;;;10.0000;;;01-Mar-2021
139618;Taurus Investor Education Pool - Unclaimed Redemption - Growth;;;10.0000;;;01-Mar-2021
139616;Taurus Unclaimed Dividend - Growth;;;13.8810;;;01-Mar-2021
139617;Taurus Unclaimed Redemption - Growth;;;13.8824;;;01-Mar-2021

Open Ended Schemes ( Equity Scheme - Multi Cap Fund )

Baroda BNP Paribas Mutual Fund
119354;Baroda BNP Paribas MULTI CAP FUND - Direct Plan - Growth Option;INF955L01AG0;;141.40;;;01-Mar-2021
119353;Baroda BNP Paribas MULTI CAP FUND - Direct Plan - IDCW Option;INF955L01AE5;;37.45;;;01-Mar-2021

## Copilot

Sure, I can help you with that! Here's a Python script that reads your text file and converts it into a one-dimensional NumPy array:

import numpy as np

# Read the file

with open('your\_file.txt', 'r') as file:

lines = file.readlines()

# Remove the header and any empty lines

data = [line.strip() for line in lines if line.strip() and not line.startswith('Column1')]

# Convert to a NumPy array

array = np.array(data)

print(array)

This script will read the file, remove any empty lines and the header, and then convert the remaining lines into a one-dimensional NumPy array. Make sure to replace 'your\_file.txt' with the actual path to your text file.

If you have any more questions or need further assistance, feel free to ask!

Source: Conversation with Copilot, 2/8/2024

1. [python - Convert a text file to a numpy array - Stack Overflow](https://stackoverflow.com/questions/22995068/convert-a-text-file-to-a-numpy-array)
2. [Import Text Files Into Numpy Arrays - GeeksforGeeks](https://www.geeksforgeeks.org/import-text-files-into-numpy-arrays/)
3. [Create an array from a .txt file in Python - CodeSpeedy](https://www.codespeedy.com/create-an-array-from-a-txt-file-in-python/)
4. [Reading and writing files — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/how-to-io.html?highlight=numpy%20any)

## You

When I apply certain python UDF on the array as above, the result is like this:

[None], [None], [None], [np.str\_('120799;Nippon India Yearly Interval Fund - Series 1 - Direct Plan - Growth Plan - Growth Option;INF204K01M39;-;16.1817;;;01-Mar-2019')], [None], [np.str\_('120798;Nippon India Yearly Interval Fund - Series 1 - Growth Plan - Growth Option;INF204K01M05;-;16.0996;;;01-Mar-2019')], [None], [None], [None], [np.str\_('143212;IIFL CAPITAL ENHANCER FUND SERIES 1-DIRECT PLAN-GROWTH;INF579M01AD3;;10.2073;;;01-Mar-2019')], [None], [None]

How to skip all [None] and the [np.str\_ which preceeds the element in the result?

## Copilot

To filter out the None values and remove the np.str\_ prefix from the elements, you can use a list comprehension. Here's an updated version of the script that processes the array accordingly:

import numpy as np

# Read the file

with open('your\_file.txt', 'r') as file:

lines = file.readlines()

# Remove the header and any empty lines

data = [line.strip() for line in lines if line.strip() and not line.startswith('Column1')]

# Convert to a NumPy array

array = np.array(data)

# Apply your UDF (assuming it returns a list with None and np.str\_ elements)

# For demonstration, let's assume the UDF is applied here and we get the result as shown

udf\_result = [None, None, None, np.str\_('120799;Nippon India Yearly Interval Fund - Series 1 - Direct Plan - Growth Plan - Growth Option;INF204K01M39;-;16.1817;;;01-Mar-2019'), None, np.str\_('120798;Nippon India Yearly Interval Fund - Series 1 - Growth Plan - Growth Option;INF204K01M05;-;16.0996;;;01-Mar-2019'), None, None, None, np.str\_('143212;IIFL CAPITAL ENHANCER FUND SERIES 1-DIRECT PLAN-GROWTH;INF579M01AD3;;10.2073;;;01-Mar-2019'), None, None]

# Filter out None values and remove np.str\_ prefix

filtered\_result = [str(item) for item in udf\_result if item is not None]

print(filtered\_result)

This script will:

1. Read the file and remove any empty lines and the header.
2. Convert the remaining lines into a NumPy array.
3. Apply your UDF (simulated here with a predefined list).
4. Filter out None values and convert np.str\_ elements to regular strings.

Let me know if you need any further assistance!

## You

I applied directly like this:
[str(item) for item in match\_pattern\_new(array,pattern) if item is not None] Where, match\_pattern\_new(array,pattern) is my UDF.

Why it's still showing same result as above? all None and np.str are still there.

## Copilot

It sounds like the issue might be with how the match\_pattern\_new function is returning its results. Let's ensure that the UDF is correctly filtering out None values and converting np.str\_ elements to regular strings.

Here's a step-by-step approach to troubleshoot and fix this:

1. **Check the UDF Output**: Print the output of match\_pattern\_new(array, pattern) to see what it returns.
2. **Ensure Correct Filtering**: Make sure the list comprehension is correctly filtering and converting the elements.

Let's start by printing the output of your UDF:

# Assuming match\_pattern\_new is your UDF

result = match\_pattern\_new(array, pattern)

print(result) # Print the result to see what it returns

If the output contains None and np.str\_ elements, then the issue is within the UDF. If not, the issue might be elsewhere.

Next, ensure the list comprehension is correctly filtering and converting the elements:

# Apply the list comprehension

filtered\_result = [str(item) for item in result if item is not None]

print(filtered\_result) # Print the filtered result

If the issue persists, you might need to adjust the UDF to ensure it returns the expected types. Here's an example of how you might modify the UDF to ensure it returns regular strings:

def match\_pattern\_new(array, pattern):

# Your existing UDF logic

# Ensure it returns regular strings instead of np.str\_

processed\_result = []

for item in array:

if some\_condition: # Replace with your actual condition

processed\_result.append(str(item))

else:

processed\_result.append(None)

return processed\_result

By ensuring the UDF returns regular strings and filtering out None values, the list comprehension should work as expected. Let me know if this helps or if you need further assistance!

## You

print result still shows:

[[None], [None], [None], [np.str\_('120799;Nippon India Yearly Interval Fund - Series 1 - Direct Plan - Growth Plan - Growth Option;INF204K01M39;-;16.1817;;;01-Mar-2019')], [None], [np.str\_('120798;Nippon India Yearly Interval Fund - Series 1 - Growth Plan - Growth Option;INF204K01M05;-;16.0996;;;01-Mar-2019')], [None], [None], [None], [np.str\_('143212;IIFL CAPITAL ENHANCER FUND SERIES 1-DIRECT PLAN-GROWTH;INF579M01AD3;;10.2073;;;01-Mar-2019')], [None], [None]]

## Copilot

It looks like the match\_pattern\_new function is returning a list of lists, with each element wrapped in a list. This is why the None values and np.str\_ elements are still present.

To fix this, you can flatten the list and then filter out the None values and convert np.str\_ elements to regular strings. Here's how you can do it:

import numpy as np

# Assuming match\_pattern\_new is your UDF

result = match\_pattern\_new(array, pattern)

print("Original result:", result) # Print the original result to see what it returns

# Flatten the list of lists

flattened\_result = [item for sublist in result for item in sublist]

# Filter out None values and convert np.str\_ elements to regular strings

filtered\_result = [str(item) for item in flattened\_result if item is not None]

print("Filtered result:", filtered\_result) # Print the filtered result

This script will:

1. Flatten the list of lists returned by match\_pattern\_new.
2. Filter out None values and convert np.str\_ elements to regular strings.

Give this a try and let me know if it resolves the issue!

## You

Combine these two list comprehension into 1:

# Flatten the list of lists

flattened\_result = [item for sublist in result for item in sublist]

# Filter out None values and convert np.str\_ elements to regular strings

filtered\_result = [str(item) for item in flattened\_result if item is not None]

## Copilot

Sure, you can combine these two list comprehensions into one. Here's how you can do it:

# Combined list comprehension to flatten, filter out None values, and convert np.str\_ elements to regular strings

filtered\_result = [str(item) for sublist in result for item in sublist if item is not None]

print(filtered\_result) # Print the filtered result

This single list comprehension will:

1. Flatten the list of lists.
2. Filter out None values.
3. Convert np.str\_ elements to regular strings.

Give this a try and let me know if it works for you!

## You

Correct! That was the issue - List of Lists.

## Copilot

Great to hear that we pinpointed the issue! Flattening the list of lists and then filtering out the None values and converting np.str\_ elements to regular strings should solve the problem.

If you have any more questions or need further assistance with your Python script or anything else, feel free to ask! 😊

## You

For converting to np array you have read the test file like this:

with open('your\_file.txt', 'r') as file:
lines = file.readlines()

But there is another option directly through numpy:

arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';')

How do you compare both in terms of performance.

## Copilot

Great question! Let's compare the two methods in terms of performance and functionality:

### 1. Using

with open('your\_file.txt', 'r') as file:

lines = file.readlines()

* **Performance**: This method is generally faster for simple file reading because it directly reads the lines into memory without additional processing.
* **Flexibility**: You have more control over how you process each line, but you need to manually handle parsing, missing values, and data types.

### 2. Using

arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';')

* **Performance**: np.genfromtxt is slightly slower because it includes additional functionality for handling missing values, data types, and more complex parsing [1].
* **Flexibility**: It provides built-in options for handling missing values, specifying data types, and more, which can simplify your code if you need these features.

### Performance Comparison:

* **Speed**: For simple file reading, open and readlines are generally faster because they do less processing [1] [2]. However, the difference might be negligible for small files.
* **Functionality**: np.genfromtxt offers more features and flexibility, making it more suitable for complex data files with missing values or mixed data types [2] [3].

### Example Use Cases:

* **Simple Reading**: Use open and readlines if you just need to read lines quickly and process them manually.
* **Complex Parsing**: Use np.genfromtxt if you need to handle missing values, specify data types, or perform more complex parsing.

Here's a quick example to illustrate both methods:

#### Using

with open('your\_file.txt', 'r') as file:

lines = file.readlines()

data = [line.strip().split(';') for line in lines if line.strip()]

array = np.array(data)

#### Using

arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';')

Both methods have their advantages, so choose the one that best fits your needs!

[1]: [Stack Overflow - Fast data reading from text file in numpy](https://stackoverflow.com/questions/35925869/fast-data-reading-from-text-file-in-numpy)
[2]: [Stack Overflow - numpy.loadtxt vs np.genfromtxt](https://stackoverflow.com/questions/51076408/np-loadtxt-vs-np-genfromtxt)
[3]: [NumPy Documentation - Importing data with genfromtxt](https://numpy.org/doc/stable/user/basics.io.genfromtxt.html)

If you have any more questions or need further assistance, feel free to ask! 😊

Source: Conversation with Copilot, 2/8/2024

1. [Fast data reading from text file in numpy - Stack Overflow](https://stackoverflow.com/questions/35925869/fast-data-reading-from-text-file-in-numpy)
2. [numpy.loadtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html)
3. [When should I use the numpy.genfromtxt instead of pandas.read\_csv to ...](https://stackoverflow.com/questions/53997674/when-should-i-use-the-numpy-genfromtxt-instead-of-pandas-read-csv-to-read-a-csv)
4. [Difference Between numpy.genfromtxt and numpy.loadtxt, and Unpack](https://stackoverflow.com/questions/20245593/difference-between-numpy-genfromtxt-and-numpy-loadtxt-and-unpack)
5. [Importing data with genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/basics.io.genfromtxt.html)
6. [python 3.x - np.loadtxt vs np.genfromtxt - Stack Overflow](https://stackoverflow.com/questions/51076408/np-loadtxt-vs-np-genfromtxt)
7. [numpy.loadtxt is way slower than open.....readlines ()](https://stackoverflow.com/questions/52232559/numpy-loadtxt-is-way-slower-than-open-readlines)

## You

Why in: arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';') delimiter has been included. We are simply reading one dimensional array?

## Copilot

The delimiter parameter in np.genfromtxt is used to specify the character that separates values in the file. Even though you're reading a one-dimensional array, the delimiter helps np.genfromtxt correctly parse the data, especially if the file contains multiple columns.

In your case, since the file has semicolon-separated values, specifying delimiter=';' ensures that each value is correctly identified and separated. If you don't specify the delimiter, np.genfromtxt might not correctly parse the data, especially if the default delimiter (whitespace) doesn't match the actual delimiter used in your file.

Here's a quick example to illustrate:

### Without Delimiter

arr = np.genfromtxt('your\_file.csv', dtype=str)

print(arr)

This might not correctly parse the semicolon-separated values, leading to incorrect data.

### With Delimiter

arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';')

print(arr)

This ensures that each semicolon-separated value is correctly parsed and included in the array.

Even if you're reading a one-dimensional array, specifying the delimiter helps ensure the data is parsed correctly. If your file truly contains only one column without any delimiters, you can omit the delimiter parameter.

If you have any more questions or need further assistance, feel free to ask! 😊

## You

using arr = .... (above command not reproducing full) gives me errors due to some line being empty or have only headings and no standard structure. How to skip lines with errors in above np.genfromtxt command?

Line #150 (got 8 columns instead of 1)
Line #151 (got 8 columns instead of 1)
Line #152 (got 8 columns instead of 1)
Line #153 (got 8 columns instead of 1)
Line #154 (got 8 columns instead of 1)
Line #155 (got 8 columns instead of 1)
Line #156 (got 8 columns instead of 1)
Line #157 (got 8 columns instead of 1)
Line #160 (got 8 columns instead of 1)
Line #161 (got 8 columns instead of 1)
Line #162 (got 8 columns instead of 1)
Line #163 (got 8 columns instead of 1)
Line #166 (got 8 columns instead of 1)
Line #167 (got 8 columns instead of 1)
Line #168 (got 8 columns instead of 1)

## Copilot

To skip lines with errors when using np.genfromtxt, you can set the invalid\_raise parameter to False. This will prevent the function from raising an exception when it encounters lines with a different number of columns than expected, and it will skip those lines instead.

Here's how you can modify your command:

arr = np.genfromtxt('your\_file.csv', dtype=str, delimiter=';', invalid\_raise=False)

This should help you avoid errors due to lines with inconsistent column counts. If you also want to handle missing values or other specific conditions, you can use additional parameters like filling\_values or missing\_values.

Give this a try and let me know if it resolves the issue! If you have any more questions or need further assistance, feel free to ask. 😊

Source: Conversation with Copilot, 2/8/2024

1. [python - Numpy genfromtxt() skip invalid lines - Stack Overflow](https://stackoverflow.com/questions/68359017/numpy-genfromtxt-skip-invalid-lines)
2. [Importing data with genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/basics.io.genfromtxt.html)
3. [numpy.genfromtxt — NumPy v1.21 Manual](https://numpy.org/doc/1.21/reference/generated/numpy.genfromtxt.html)
4. [Reading and writing files — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/how-to-io.html?highlight=numpy%20genfromtxt)
5. [np.genfromtxt() Function in Python [5 Examples] - Python Guides](https://pythonguides.com/python-numpy-genfromtxt/)
6. [numpy.genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html)
7. [How to control genfromtxt to read rows specified?](https://stackoverflow.com/questions/25925348/how-to-control-genfromtxt-to-read-rows-specified)
8. [numpy.genfromtxt skips/ignores last line in long tsv file](https://stackoverflow.com/questions/23082561/numpy-genfromtxt-skips-ignores-last-line-in-long-tsv-file)

## You

Actually I want lines with eight columns and not the ones with headings only or empty. Is there any way to have converse in above command?

## Copilot

To achieve this, you can use np.genfromtxt with a custom function to filter out lines that don't match the desired number of columns. Here's how you can do it:

1. **Read the file** and filter out lines with exactly 8 columns.
2. **Use** np.genfromtxt to process the filtered lines.

Here's a complete example:

import numpy as np

# Custom function to filter lines with exactly 8 columns

def filter\_lines(file\_path, delimiter=';', num\_columns=8):

with open(file\_path, 'r') as file:

lines = file.readlines()

filtered\_lines = [line for line in lines if len(line.strip().split(delimiter)) == num\_columns]

return filtered\_lines

# Path to your file

file\_path = 'your\_file.csv'

# Filter lines

filtered\_lines = filter\_lines(file\_path)

# Write filtered lines to a temporary file

with open('filtered\_file.csv', 'w') as file:

file.writelines(filtered\_lines)

# Use np.genfromtxt to read the filtered file

arr = np.genfromtxt('filtered\_file.csv', dtype=str, delimiter=';')

print(arr)

This script will:

1. Read the file and filter out lines that don't have exactly 8 columns.
2. Write the filtered lines to a temporary file.
3. Use np.genfromtxt to read the filtered file into a NumPy array.

This way, you can ensure that only lines with 8 columns are processed. Let me know if this works for you or if you need any further assistance! 😊

## You

Instead of so many commands to derive 'filtered\_file.csv' can this be shortened so that in np.genfromtxt, it only considers lines where delimiters ";" are present and skips all others?

## Copilot

You can achieve this by using a generator to filter the lines before passing them to np.genfromtxt. This way, you can directly skip lines that don't match the desired format. Here's how you can do it:

import numpy as np

# Custom generator to filter lines with exactly 8 columns

def filter\_lines(file\_path, delimiter=';', num\_columns=8):

with open(file\_path, 'r') as file:

for line in file:

if len(line.strip().split(delimiter)) == num\_columns:

yield line

# Path to your file

file\_path = 'your\_file.csv'

# Use np.genfromtxt with the generator

arr = np.genfromtxt(filter\_lines(file\_path), dtype=str, delimiter=';')

print(arr)

This script will:

1. Use a generator function filter\_lines to read the file line by line and yield only those lines that have exactly 8 columns.
2. Pass the generator directly to np.genfromtxt, which will process only the filtered lines.

This approach avoids the need for intermediate files and keeps the code concise. Let me know if this works for you or if you need any further assistance! 😊

Source: Conversation with Copilot, 2/8/2024

1. [python - Numpy genfromtxt() skip invalid lines - Stack Overflow](https://stackoverflow.com/questions/68359017/numpy-genfromtxt-skip-invalid-lines)
2. [Importing data with genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/basics.io.genfromtxt.html)
3. [numpy.genfromtxt — NumPy v1.21 Manual](https://numpy.org/doc/1.21/reference/generated/numpy.genfromtxt.html)
4. [Reading and writing files — NumPy v2.0 Manual](https://numpy.org/doc/stable/user/how-to-io.html?highlight=numpy%20genfromtxt)
5. [np.genfromtxt() Function in Python [5 Examples] - Python Guides](https://pythonguides.com/python-numpy-genfromtxt/)
6. [Difference Between numpy.genfromtxt and numpy.loadtxt, and Unpack](https://stackoverflow.com/questions/20245593/difference-between-numpy-genfromtxt-and-numpy-loadtxt-and-unpack)
7. [Fast data reading from text file in numpy - Stack Overflow](https://stackoverflow.com/questions/35925869/fast-data-reading-from-text-file-in-numpy)
8. [numpy.loadtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html)
9. [When should I use the numpy.genfromtxt instead of pandas.read\_csv to ...](https://stackoverflow.com/questions/53997674/when-should-i-use-the-numpy-genfromtxt-instead-of-pandas-read-csv-to-read-a-csv)
10. [python 3.x - np.loadtxt vs np.genfromtxt - Stack Overflow](https://stackoverflow.com/questions/51076408/np-loadtxt-vs-np-genfromtxt)
11. [numpy.loadtxt is way slower than open.....readlines ()](https://stackoverflow.com/questions/52232559/numpy-loadtxt-is-way-slower-than-open-readlines)
12. [How to control genfromtxt to read rows specified?](https://stackoverflow.com/questions/25925348/how-to-control-genfromtxt-to-read-rows-specified)
13. [numpy.genfromtxt skips/ignores last line in long tsv file](https://stackoverflow.com/questions/23082561/numpy-genfromtxt-skips-ignores-last-line-in-long-tsv-file)
14. [numpy.genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html)
15. [Importing data with genfromtxt — NumPy v2.0 Manual](https://numpy.org/doc/stable//user/basics.io.genfromtxt.html)