Assignment 2 - Numpy Array Operations

5 Must Known Numpy Array Functions

Here we are going to explore 5 most important numpy functions to play with textualfile.

- `numpy.asarray` (Convert the input to a chararray, copying the data only if necessary.)
- `numpy.genfromtxt` (Load data from a text file, with missing values handled as specified.)
- `numpy.savetxt` (Save an array to a text file.)
- `numpy.fromfunction` (Construct an array by executing a function over each coordinate..)
- `numpy.loadtxt`(Load data from a text file. when no data is missing.)

```
!pip install jovian --upgrade -q
```

```
import jovian
```

```
jovian.commit(project='numpy-array-operations')
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

Let's begin by importing Numpy and listing out the functions covered in this notebook.

```
import numpy as np
```

List of functions explained

- `np.asarray` (this function is used to convert the input to a chararray, copying the data only if necessary.)
- `np.genfromtxt` (this function is used to load data from a text file, with missing values handled as specified.)
- `np.savetxt` (this function is used save an array to a text file.)
- `np.fromfile` (this function is used construct an array from data in a text or binary file.)
- `np.loadtxt`(this function is used load data from a text file. when no data is missing.)

Function 1 - np.asarray

The `np.asarray()` function is used to convert input data into a `numpy` array.

the genral syntax for the np.asarray() function:-

`np.asarray(a, dtype=None, order=None)`

a : Input data, you want to convert to a NumPy array.

dtype : Optional data type for the array elements.

order : Optional

```
# Example 1 - working (Converting a List to a NumPy Array
jovian_fee = [12,23,43,54,76,89]
jovian_array = np.asarray(jovian_fee)
print(jovian_array)
```

```
[12 23 43 54 76 89]
```

Here's np.asarray() function converts a Python list jovian_fee into a numpy array jovian_array.

```
# Example 2 - working(Converting tuple into numpy array)
import numpy as np
jovian_tuple = (10, 20, 30, 40)
jovian_array = np.asarray(jovian_tuple)

print(jovian_array)
print(type(jovian_array))
```

```
[10 20 30 40]
<class 'numpy.ndarray'>
```

here np.asarray() function is used to convert a tuple into a NumPy array.

```
# Example 3 - breaking (when you trying to convert input data into numpy array that has

import numpy as np
arr = np.asarray([1, 2, 3], shape=(2, 3))
print(arr)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_77/1647507886.py in <module>
      2
      3 import numpy as np
----> 4 arr = np.asarray([1, 2, 3], shape=(2, 3))
      5 print(arr)

TypeError: asarray() got an unexpected keyword argument 'shape'
```

The input data is not an object of the correct shape. so specify shape argument proper shape of the output array.

Use this function when you want to form an array from user input

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

Function 2 - np.genfromtxt

`np.genfromtxt` function used for loading data from text files into NumPy arrays. usually used when you have tabular data stored in text files (such as CSV files) and you want to create NumPy arrays from that data for analysis or manipulation.

syntax for np.genfromtxt() function:-

```
np.genfromtxt(fname, delimiter=',', dtype=None, names=True)
```

numpy_array = np.genfromtxt(fname, delimiter=',', dtype=None, names=True)

`fname` : file name that containing the data.

`delimiter` : the delimiter character or string that separates the values in the text file.

`dtype` : data type to which the elements should be converted

`names` : If True, the first row of the data is treated as column names, and you can access columns by these names.

```
# Example 1 - working(Data Loading and Preprocessing)
#Let's suppose i have csv file jovian.csv in my current working directry
import numpy as np
import random
import pandas as pd

jovian_data = {
    'Name':['abhsihek', 'vivek', 'ram'],
    'age':[23,34,12],
    'fee':[23000, 22000, 21000]}

import pandas as pd
df = pd.DataFrame(jovian_data)
df.to_csv('jovian.csv', index=False)

data = np.genfromtxt('jovian.csv', delimiter=',', names=True, dtype=None, encoding=None
print(data)
```

```
[('abhsihek', 23, 23000) ('vivek', 34, 22000) ('ram', 12, 21000)]
```

Here in this above example we load the data jovian.csv into a NumPy array by specifies that the data is comma-separated, names=True indicates that the first row contains column names, dtype=None and encoding=None

```
# Example 2 - Missing Data Handling:
#Let's suppose there is some missing value in the csv file jovian_miss.csv in the curre
import numpy as np
import random
import pandas as pd
import os

"""
Name,Age,fees
John,25,50000
Jane,,60000
Mike,30,
"""

import numpy as np

missing_values = ['', 'NA', 'N/A']
filling_values = [np.nan, np.nan, np.nan]  # Using np.nan to represent missing values

data = np.genfromtxt(r"C:\VsCode\jovian_miss.csv", delimiter=',', names=True, dtype=Non

print(data)
```

here above, we np.nan is used as a placeholder for missing values.The missing_values parameter is defined if these values would be showd in data then we fill values defined in filling_values variable at missing place.

```
# Example 3 - breaking (to illustrate when it breaks)
import numpy as np
import random
import pandas as pd

jovian_data = {
    'Name':['abhsihek', 'vivek', 'ram'],
    'age':[23],
    'fee':[23000, 22000]}

import pandas as pd
df = pd.DataFrame(jovian_data)
df.to_csv('jovian.csv', index=False)

missing_values = ['', 'NA', 'N/A']
filling_values = [0, 0, 0]
data = np.genfromtxt('jovian.csv', delimiter=',', names=True, dtype=None, encoding=None
print(data)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_77/3250006807.py in <module>
     10
     11 import pandas as pd
---> 12 df = pd.DataFrame(jovian_data)
```

```
     13 df.to_csv('jovian.csv', index=False)
     14

/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py in __init__(self, data,
index, columns, dtype, copy)
    612         elif isinstance(data, dict):
    613             # GH#38939 de facto copy defaults to False only in non-dict cases
--> 614             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy,
typ=manager)
    615         elif isinstance(data, ma.MaskedArray):
    616             import numpy.ma.mrecords as mrecords

/opt/conda/lib/python3.9/site-packages/pandas/core/internals/construction.py in
dict_to_mgr(data, index, columns, dtype, typ, copy)
    462         # TODO: can we get rid of the dt64tz special case above?
    463
--> 464     return arrays_to_mgr(
    465         arrays, data_names, index, columns, dtype=dtype, typ=typ,
consolidate=copy
    466     )

/opt/conda/lib/python3.9/site-packages/pandas/core/internals/construction.py in
arrays_to_mgr(arrays, arr_names, index, columns, dtype, verify_integrity, typ,
consolidate)
    117         # figure out the index, if necessary
    118         if index is None:
--> 119             index = _extract_index(arrays)
    120         else:
    121             index = ensure_index(index)

/opt/conda/lib/python3.9/site-packages/pandas/core/internals/construction.py in
_extract_index(data)
    633             lengths = list(set(raw_lengths))
    634             if len(lengths) > 1:
--> 635                 raise ValueError("All arrays must be of the same length")
    636
    637             if have_dicts:

ValueError: All arrays must be of the same length
```

Here in this above example error occured due to All arrays not same so it must be necessary to have same length of array

When you want to load data from text use this np.genfromtxt() as well as want handle missing values from your text data.

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

Function 3 - np.savetxt

sometime it would be necessary to save our array file for further working or manipulating so for this your concern np.savetxt for you which is used to save an array to a text file.

Where:-

General syntax for np.savetxt function is :-

```
numpy.savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None)
```

fname: The file name or file object where the data will be saved.

X: The data array to be saved. It can be a 1D or 2D NumPy array.

delimiter: A character that separates values in the output.

newline: string used to separate lines in the output file.

header: String that will be written at the beginning of the file..

footer: String that will be written at the end of the file.

fmt: The format string used for formatting the data.

comments: The character(s) used to indicate comments in the header.

encoding: The text encoding to be used when writing the file.

```
# Example 1 - (Saving numpy array to CSV file)
import numpy as np

arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

np.savetxt('abhishek.csv', arr, delimiter=',')
```

In this example we created a NumPy array and then save it to a CSV file using np.savetxt where we also specified that the values should be separated by commas "," in the CSV file.

```
# Example 2 - (Saving NumPy Array with Custom Formatting)
import numpy as np

# Create the NumPy array
```

```
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

# Save the array with custom formatting
np.savetxt('abhi.csv', arr, fmt='%.2f', delimiter='\t')
```

Explanation about example : here we have saved numpy array by specifying number format should be 2 decimal places and delimter is tab

```
# Example 3 - breaking (it breaks when array has not same data types)
import numpy as np

arr = np.array([1, 2.5, 3, '4.9'])
np.savetxt('my_data.txt', arr, fmt='%d')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in savetxt(fname, X, fmt,
delimiter, newline, header, footer, comments, encoding)
   1432                 try:
-> 1433                     v = format % tuple(row) + newline
   1434                 except TypeError as e:

TypeError: %d format: a number is required, not numpy.str_

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_46/3958252033.py in <module>
      3
      4 arr = np.array([1, 2.5, 3, '4.9'])
----> 5 np.savetxt('my_data.txt', arr, fmt='%d')

<__array_function__ internals> in savetxt(*args, **kwargs)

/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in savetxt(fname, X, fmt,
delimiter, newline, header, footer, comments, encoding)
   1433                     v = format % tuple(row) + newline
   1434                 except TypeError as e:
-> 1435                     raise TypeError("Mismatch between array dtype ('%s') and "
   1436                                     "format specifier ('%s')"
   1437                                     % (str(X.dtype), format)) from e

TypeError: Mismatch between array dtype ('<U32') and format specifier ('%d')
```

Due to having diffrent data type to an array, this methods has given error we can simply fix it by defining proper data types

```
# here we can set format string of characters '%s'
import numpy as np

arr = np.array([1, 2.5, 3, '4.9'])
np.savetxt('my_data.txt', arr, fmt='%s')
```

```
np.loadtxt('my_data.txt')
```

```
array([1. , 2.5, 3. , 4.9])
```

when you want to save your data in your system as a text with custom foramtting use np.save method.

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

Function 4 - numpy.vectorize(Returns an object that acts like pyfunc, but takes arrays as input.)

numpy.vectorize : It is used to create a "vectorized" version of a Python function, allowing you to apply that function element-wise to NumPy arrays.

numpy.vectorize by Define a vectorized function which takes a nested sequence of objects or numpy arrays as inputs and returns a single numpy array or a tuple of numpy arrays. The vectorized function evaluates pyfunc over successive tuples of the input arrays like the python map function, except it uses the broadcasting rules of numpy.

Genral syntax

> **numpy.vectorize(pyfunc=np._NoValue, otypes=None, doc=None, excluded=None, cache=False, signature=None)**

> pyfunc: The Python function you want to apply to each element of an array.

> otypes (optional): Specifies the data types for the output arrays, ensuring consistent data types.

> doc (optional): Allows you to add a docstring to the new vectorized function.

> excluded (optional): Lets you exclude certain arguments when vectorizing the function.

> cache (optional): If True, caches results for improved performance with repeated inputs.

signature (optional): An advanced option for precise control over function application.

```
# Example 1 - (Element-Wise Function on a NumPy Array)
# let's suppose we have a custom function mera function we can apply to every element o
import numpy as np

# custom Python function
def mera_function(x):
    return x ** 2

#  Define a vectorized function
squared_arr = np.vectorize(mera_function)

my_arr = np.array([1, 2, 3, 4, 5])
result = squared_arr(arr)
result
```

```
array([ 1,  4,  9, 16, 25])
```

Above custom function is mera_function where we defined vectorized function as squared_arr which takes a numpy arrays "my_arr" as inputs and returns a single numpy array.

The vectorized function evaluates pyfunc over successive tuples of the input arrays like the python map function, except it uses the broadcasting rules of numpy.

```
# Example 2 - (Handling Missing Values or NaNs)
import numpy as np

# custom function for handle NaNs
def replace_nan(x, replacement):
    return x if not np.isnan(x) else replacement

# vectorized version of the function
vectorized_replace_nan = np.vectorize(replace_nan)

# An array with NaN values
arr = np.array([1.0, 2.0, np.nan, 4.0, np.nan])

# Replace NaNs with a 0
reslt = vectorized_replace_nan(arr, replacement=0)

print(reslt)
```

```
[1. 2. 0. 4. 0.]
```

here we use np.vectorize to handle missing values NaNs in a NumPy array "arr" by apply custom function "replace_nan"

```
# Example 3 - breaking (it breaks when we have not proper shape)
import numpy as np

def custom_function(x, y):
    return x + y

vectorized_func = np.vectorize(custom_function)

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5])

# Apply the vectorized function to the arrays with different shapes
result = vectorized_func(arr1, arr2)

result
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_46/3391249900.py in <module>
     11
     12 # Apply the vectorized function to the arrays with different shapes
---> 13 result = vectorized_func(arr1, arr2)
     14
     15 result

/opt/conda/lib/python3.9/site-packages/numpy/lib/function_base.py in __call__(self,
*args, **kwargs)
   2111             vargs.extend([kwargs[_n] for _n in names])
   2112
-> 2113         return self._vectorize_call(func=func, args=vargs)
   2114
   2115     def _get_ufunc_and_otypes(self, func, args):

/opt/conda/lib/python3.9/site-packages/numpy/lib/function_base.py in
_vectorize_call(self, func, args)
   2195                       for a in args]
   2196
-> 2197             outputs = ufunc(*inputs)
   2198
   2199             if ufunc.nout == 1:

ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

it breaks bcz arr1 and arr2 have different shapes array must have same shape we can transform array by filling value.

np.vectorize is a good way to apply custom Python functions to each element of NumPy arrays and often most helpful for element-wise operations without the need for writing custom ufuncs.

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

'https://jovian.com/ay29020/numpy-array-operations'

Function 5 - - numpy.loadtxt(Load data from a text file. when no data is missing.)

numpy.loadtxt method is the flexible way of loading data from text file.

```
# Example 1 - working(loadinging NumPy arrays from text files)
import numpy as np
arr = np.arange(10).reshape(5,2)
headers = "Column 1, Column 2"
np.savetxt('raw', arr, delimiter='\t', fmt='%.2f', header='headers')
gappu_mark = np.loadtxt('raw', delimiter='\t')
gappu_mark
```

```
array([[0., 1.],
       [2., 3.],
       [4., 5.],
       [6., 7.],
       [8., 9.]])
```

Here we make an identity array as arr and then save as text file raw in our local after that we loaded this data from text file raw

```
# Example 2 - working(skip headers from data)
import pandas as pd
import numpy as np
raw_data = {
    'abhishek':[1,2,3,4],
    'vivek':[5,6,7,8]
}
df = pd.DataFrame(raw_data)
df.to_csv('my_data.txt',sep ='\t', header = ['a','b'], float_format='%.2f')
np.loadtxt('my_data.txt',skiprows=1)
# array = df.values
```

```
array([[0., 1., 5.],
       [1., 2., 6.],
       [2., 3., 7.],
       [3., 4., 8.]])
```

here

```
# Example 3 - breaking (to illustrate when it breaks)
import pandas as pd
import numpy as np
raw_data = {
```

```
    'abhishek':[1,2,3,4],
    'vivek':[5,6,7,8]
}
df = pd.DataFrame(raw_data)
df.to_csv('my_data.txt',sep ='\t', header = ['a','b'], float_format='%.2f')
np.loadtxt('my_data.txt')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_47/1781759793.py in <module>
      9 df = pd.DataFrame(raw_data)
     10 df.to_csv('my_data.txt',sep ='\t', header = ['a','b'], float_format='%.2f')
---> 11 np.loadtxt('my_data.txt')

/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in loadtxt(fname, dtype,
comments, delimiter, converters, skiprows, usecols, unpack, ndmin, encoding, max_rows,
like)
   1144         # converting the data
   1145         X = None
-> 1146         for x in read_data(_loadtxt_chunksize):
   1147             if X is None:
   1148                 X = np.array(x, dtype)

/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in read_data(chunk_size)
    995
    996             # Convert each value according to its column and store
--> 997             items = [conv(val) for (conv, val) in zip(converters, vals)]
    998
    999             # Then pack it according to the dtype's nesting

/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in <listcomp>(.0)
    995
    996             # Convert each value according to its column and store
--> 997             items = [conv(val) for (conv, val) in zip(converters, vals)]
    998
    999             # Then pack it according to the dtype's nesting

/opt/conda/lib/python3.9/site-packages/numpy/lib/npyio.py in floatconv(x)
    732         if '0x' in x:
    733             return float.fromhex(x)
--> 734         return float(x)
    735
    736     typ = dtype.type

ValueError: could not convert string to float: 'a'
```

here's we getting ValueError due to my_data.txt has column headers when np.loads tries to convert the headers ('a' and 'b') to floats, it raises a ValueError so for overcome this issue we can use `skiprows` parameter

numpy.loadtxt method is very efficient to load data from a text file.it's similar to np.genfromtxt. but does not offer missing values handling funtionality like np.genfromtxt.

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

Conclusion

Summarize what was covered in this notebook, and where to go next

Reference Links

- Numpy official tutorial : https://numpy.org/doc/stable/reference/routines.html

```
jovian.commit()
```

```
[jovian] Updating notebook "ay29020/numpy-array-operations" on https://jovian.com
[jovian] Committed successfully! https://jovian.com/ay29020/numpy-array-operations
```

```
'https://jovian.com/ay29020/numpy-array-operations'
```

```

```

```

```