Data wrangling in Python - Data wrangling with NumPy - 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Data Wrangling with NumPy: Topic introduction

In this part of the course, we will cover the following concepts:

- NumPy use cases and object types
- NumPy array manipulation

Warm up

- Have a look at the dataset to the right
- What **questions** do you have about it?
- What might make it **difficult** to work with?
- **Share your thoughts** in the chat or aloud

Name	Phone	Birth Date	State
John, Smith	445-881-4478	August 12, 1989	Maine
Jennifer Tal	+1-189-456-4513	11/12/1965	Tx
Gates, Bill	(876)546-8165	June 15, 72	Kansas
Alan Fitch	5493156648	2-6-1985	Oh
Jacob Alan	156-4896	January 3	Alabama

Source: ***wikipedia***

Importance of clean data

- In today's session, we're going to talk about how to **prepare and organize data**
- The process is also known as **data cleaning** and **data wrangling**

(a) Dirty Data

id	title	pub_year	citation _count
*t*1	CrowdDB	11	18
*t*2	TinyDB	2005	1569
*t*3	YFilter	Feb, 2002	298
*t*4	Aqua		106
*t*5	DataSpace	2008	107
*t*6	CrowdER	2012	1
*t*7	Online Aggr.	1997	687
...	...	...	...
*t*10000	YFilter - ICDE	2002	298

(b) Cleaned Sample

id	title	pub_year	citation _count	#dup
*t*1	CrowdDB	**2011**	**144**	**2**
*t*2	TinyDB	2005	1569	1
*t*3	YFilter	**2002**	298	**2**
*t*4	Aqua	**1999**	106	1
*t*5	DataSpace	2008	107	1
*t*6	CrowdER	2012	**34**	1
*t*7	Online Aggr.	1997	687	**3**

Source: *Semantic Scholar*

Module completion checklist

In this module, we will explore **NumPy objects**

Objective	Complete
Illustrate NumPy objects in Python	
Explore NumPy array data types and implement more NumPy objects	

Data wrangling and exploration

- Remember, a data scientist must be able to:

1. **Wrangle** the data (gather, clean, and sample data to get a suitable dataset)
2. **Manage** the data for easy access by the organization
3. **Explore** the data to generate a hypothesis

- We will learn how to use one of the two powerful Python libraries, **NumPy and Pandas**, that will help us achieve these goals!

Introduction to NumPy

- NumPy is widely used in machine learning and scientific computing due to its basic core data structure: **array**
- It is also widely used in combination with `matplotlib` and other plotting libraries to create graphs
- NumPy's array functions are similar to those available for vectors in MATLAB and R

NumPy

Scipy.org

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

Creating arrays

- There are multiple ways to create a NumPy array
- One of the easiest is to make it from a `list` and using NumPy's `array()` function
- To use the `array()` function, we need to import `numpy`
- When writing code, we usually want to **import all packages** needed for the program at the beginning

```
# Import numpy as 'np' sets 'np' as the shortcut/alias.
import numpy as np

# Create an array from a list.
arr = np.array([17, -10, 16.8, 11])
print(arr)

# Check the type of the object.
```

```
[ 17.   -10.    16.8   11. ]
```

```
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

Data type in arrays

- NumPy arrays have a property called `dtype` which records the data type of the array's members
- NumPy arrays are required to have the **same data type**
- That is why they are called **atomic** data structures (i.e. structures that allow a single data type)

```
# Check the data type stored in the array.
print(arr.dtype)
```

```
float64
```

Using ndarray

- The most important data type that NumPy provides is the “n-dimensional array,” `ndarray`
- An `ndarray` is similar to a Python list in which all members have the same data type
- We create it using `np.array()`

```
x = np.array([3, 19, 7, 11])
print(x)
```

```
[ 3 19  7 11]
```

Documentation for ndarray

- Each package in Python, like NumPy, has ***documentation*** for each function within

array

A homogeneous container of numerical elements. Each element in the array occupies a fixed amount of memory (hence homogeneous), and can be a numerical element of a single type (such as float, int or complex) or a combination (such as `(float, int, float)`). Each array has an associated data-type (or `dtype`), which describes the numerical type of its elements:

```
>>> x = np.array([1, 2, 3], float)

>>> x
array([ 1.,  2.,  3.])

>>> x.dtype # floating point number, 64 bits of memory per element
dtype('float64')


# More complicated data type: each array element is a combination of
# and integer and a floating point number
>>> np.array([(1, 2.0), (3, 4.0)], dtype=[('x', int), ('y', float)])
array([(1, 2.0), (3, 4.0)],
      dtype=[('x', '<i4'), ('y', '<f8')])
```

Fast element-wise operations, called a ufunc, operate on arrays.

Building an array with linspace

- Another function we can use to build an array is `np.linspace`

```
y = np.linspace(-2, -1, 25)
print(y)
```

```
[-2.         -1.95833333 -1.91666667
-1.875      -1.83333333 -1.79166667
 -1.75       -1.70833333 -1.66666667
-1.625      -1.58333333 -1.54166667
 -1.5        -1.45833333 -1.41666667
-1.375      -1.33333333 -1.29166667
 -1.25       -1.20833333 -1.16666667
-1.125      -1.08333333 -1.04166667
 -1.        ]
```

- This function will return 25 numbers between -2 and -1

numpy.linspace

numpy.linspace(*start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0*) [source]

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*].

The endpoint of the interval can optionally be excluded.

Changed in version 1.16.0: Non-scalar *start* and *stop* are now supported.

Alternative ways of accessing functions

- If you are only going to use a handful of functions from a library, you can access functions in the following way:

```
from numpy import array, linspace
x = array([0.01, 0.45, -0.3])
y = linspace(0, 1, 50)
```

- With this syntax, we can use `array` or `linspace` without the `np.` prefix

Module completion checklist

Objective	Complete
Illustrate NumPy objects in Python	✔
Explore NumPy array data types and implement more NumPy objects	

NumPy array data types

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
`intc`	Identical to C `int` (normally `int32` or `int64`)
`intp`	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
`int8`	Byte (-128 to 127)
`int16`	Integer (-32768 to 32767)
`int32`	Integer (-2147483648 to 2147483647)
`int64`	Integer (-9223372036854775808 to 9223372036854775807)

NumPy array data types (cont’d)

Data type	Description
`uint8`	Unsigned integer (0 to 255)
`uint16`	Unsigned integer (0 to 65535)
`uint32`	Unsigned integer (0 to 4294967295)
`uint64`	Unsigned integer (0 to 18446744073709551615)
`float_`	Byte (-128 to 127)
Shorthand for `float64`	Integer (-32768 to 32767)
`float16`	Integer (-2147483648 to 2147483647)
`int64`	Integer (-9223372036854775808 to 9223372036854775807)

Arrays vs. lists

- Unlike lists, NumPy arrays can hold values of only a **single data type**
- This makes arrays much more powerful for vectorized complex manipulations
- Let's see what happens if we try to create an array from a list of mixed data types

```
mixed_array = np.array([1, 2, "apple", "XYZ", 5.5])
print(mixed_array)
```

```
['1' '2' 'apple' 'XYZ' '5.5']
```

- **Question:** What do you think the data type of `mixed_array` will be?

Arrays vs lists (cont'd)

- The answer is **<U21**, which is a data type for Unicode strings

```
print(mixed_array.dtype)
```

```
<U21
```

- This means that all values in the initial list are **cast** into, or interpreted as, **string data** to maintain homogeneity
- Similarly, creating an array from a list of mixed integer and float values changes all elements to the **float** data type

```
mixed_array = np.array([3, 12, 5.56])
print(mixed_array)
```

```
[ 3.    12.     5.56]
```

```
print(mixed_array.dtype)
```

```
float64
```

- You can read more about NumPy data types ***here***

Arrays from sequences

- We can also create an array that contains a **sequence**, like a series of numbers
- To create the range of numbers of 0 to 50, use the `arange` command

numpy.arange

numpy.arange([*start*,]*stop*, [*step*,]*dtype=None*)

Return evenly spaced values within a given interval.

Values are generated within the half-open interval `[start, stop)` (in other words, the interval including *start* but excluding *stop*). For integer arguments the function is equivalent to the Python built-in *range* function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use numpy.linspace for these cases.

Arrays from sequences (cont'd)

```
rng = np.arange(0, 51)
print(rng)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50]
```

- The last number in the range is one less than the value you provided, so we provide 51 to ensure that the last value is 50

Arrays from sequences - using a step size

- To create a sequence of only some of the values in a range,like only the even values, we can specify a **step size**
- Let’s see what happens if we increase the step size from the default 1

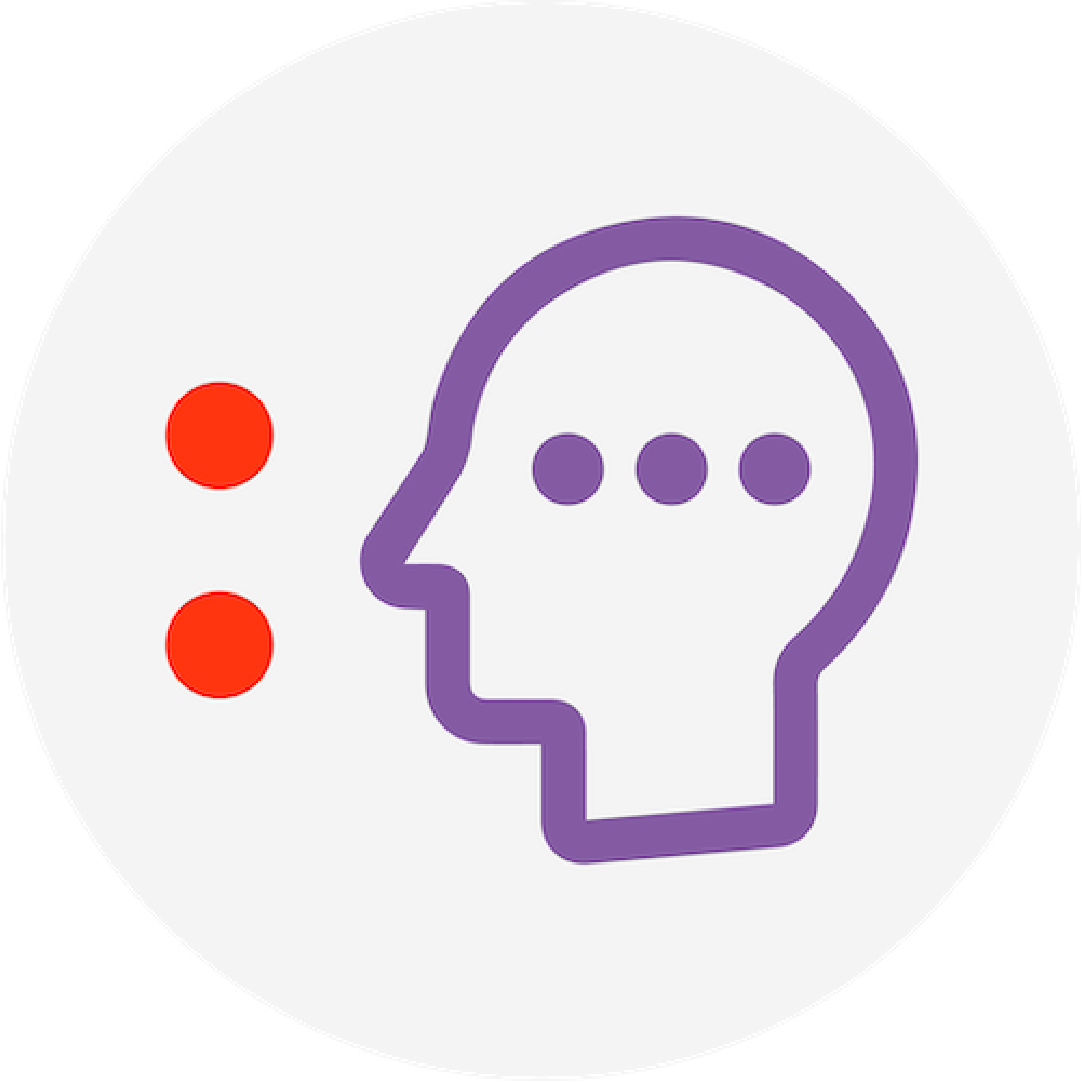
```
evens = np.arange(0, 23, 2)
print(evens)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20 22]
```

```
quarters = np.arange(0, 1, .25)  #<- contains 0 to 0.75
print(quarters)
```

```
[0.   0.25 0.5  0.75]
```

Knowledge check



Link: ***https://forms.gle/kM4mV92NNiSMFLzbA***

Module completion checklist

Objective	Complete
Illustrate NumPy objects in Python	✔
Explore NumPy array data types and implement more NumPy objects	✔

Congratulations on completing this module!

You are now ready to try Tasks 1-6 in the Exercise for this topic