STATS 507
Data Analysis in Python

Week6-1: More on NumPy

Dr. Xian Zhang

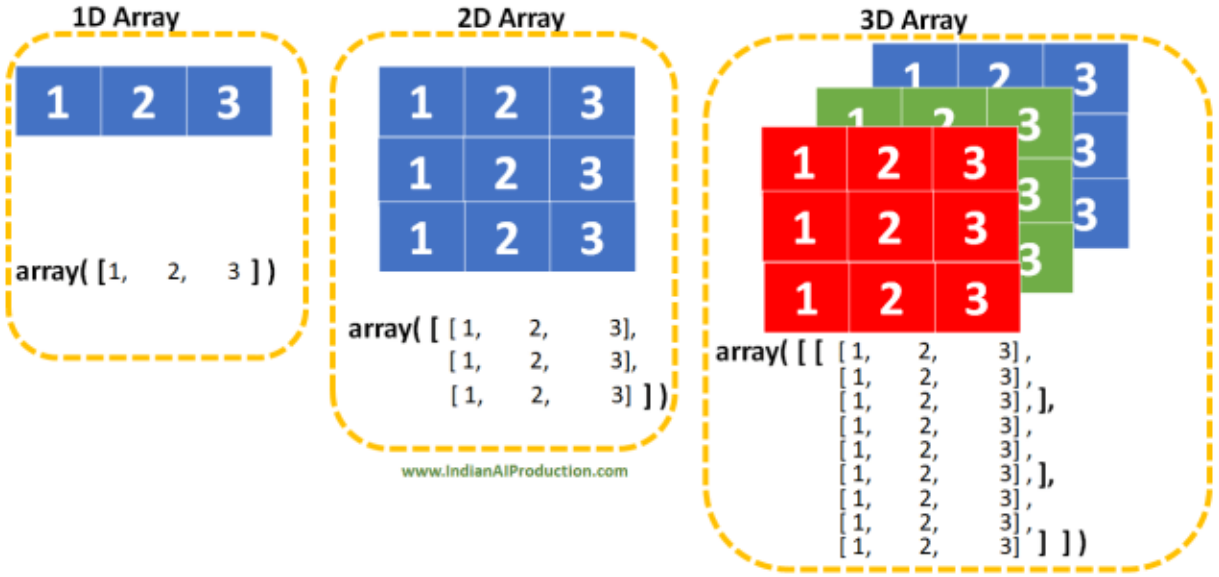
Adapted from slides by Professor Jeffrey Regier

NumPy EcoSystem…

Recap: NumPy as numerical Python

One of the key data type of NumPy is its **N-dimensional array object** (also referred as: array, NumPy array, np.ndarray).

- A NumPy array is a grid of values, **all of the same type**
- *Rank* of the array: the dimension of the arrays
- *Shape*: a tuple of integers giving the size of the array along each dimension



Recap: Creating NumPy Array

1) Converting Python sequences to NumPy arrays (lists and tuples)

```
# From a list
arr1 = np.array([1, 2, 3, 4, 5], dtype = 'uint')
print(type(arr1))
# From a tuple
arr2 = np.array((1, 2, 3, 4, 5), dtype = 'uint')
print(type(arr2))
```

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

2) Using NumPy Arrays functions

```
# Create an array of zeros
zeros_arr = np.zeros(5)  # [0. 0. 0. 0. 0.]
# Create an array of ones
# [[1. 1. 1.]
#  [1. 1. 1.]]
ones_2d = np.ones((2, 3))
# Create an array with a range of values
range_arr = np.arange(0, 10, 2)  # [0 2 4 6 8]
# Create an array with evenly spaced values
linspace_arr = np.linspace(0, 1, 5)  # [0.   0.25 0.5  0.75 1.  ]
```

3) Create NumPy array based on the properties of existing arrays

```
import numpy as np

# Create a sample array
sample = np.array([[1, 2, 3], [4, 5, 6]])

# Create a new array with the same shape as sample, filled with 7
full_like_arr = np.full_like(sample, 7)
print(full_like_arr)
```

```
[[7 7 7]
 [7 7 7]]
```

```
# Can be replaced with ones_like
zeros_like_arr = np.zeros_like(sample)
print(zeros_like_arr)
```

```
[[0 0 0]
 [0 0 0]]
```

```
# Get array attributes
print(b.ndim) # dimension of the array
print(b.shape) # shape of the array
print(b.dtype) # data type
print(b.size) # no. of elements
```

Return a tuple

Recap: NumPy Array operations

Array indexing

Indexing/ slicing for each dimension of the array.

```
import numpy as np
a = np.array([[1,2], [3,4], [5,6]])
```

```
a[:2,1:]
```

```
array([[2],
       [4]])
```

Integer array indexing (arbitrary arrays)

```
a[[0,1], [1,0]]
```

```
array([2, 3])
```

Boolean array indexing for conditional selection.

```
a[a>2]
```

```
array([3, 4, 5, 6])
```

Vector operations

Basic mathematical functions operate **elementwise** on arrays:

1) Operator overloads
2) Function in NumPy

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

Vector operations (dot, inner, outer product) is method of array objects available both as a

1) function in the NumPy
2) as an instance method

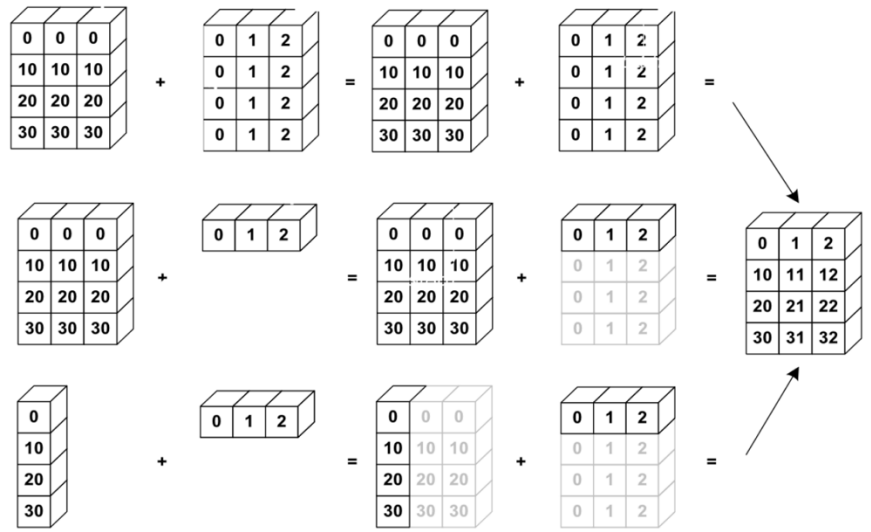
```
# Matrix / vector product;
print(x.dot(v))
print(np.dot(x, v))
```

Broadcasting

NumPy works with arrays of **different shapes** when performing **arithmetic** operations.

The arrays can be broadcast together only if they are compatible in all dimensions.

Equal size or one of them is 1



1. More on array manipulation

2. More on ufuncs

3. Statistics in NumPy

More on broadcasting

Can I performing **arithmetic** operations (+)?

```
my_matrix = np.array([[1, 2, 3],
                      [4, 5, 6]])
my_vector = np.array([10, 20])
my_matrix + my_vector
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[134], line 4
      1 my_matrix = np.array([[1, 2, 3],
      2                       [4, 5, 6]])
      3 my_vector = np.array([10, 20])
----> 4 my_matrix + my_vector

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

The arrays can **NOT** be broadcast together only if they are not compatible in all dimensions.

What should we do?

More on broadcasting

`np.newaxis` and `None` are used to **add a new dimension** to an existing array.

```
# Expanding vector dimensions using np.newaxis for compatibility
my_vector = np.array([10, 20])
print(my_vector.shape)
my_vector = my_vector[:, np.newaxis]
print(my_vector.shape)
my_matrix + my_vector
```

```
(2,)
(2, 1)
array([[11, 12, 13],
       [24, 25, 26]])
```

`np.newaxis` and `None` are interchangeable in NumPy

```
# Expanding vector dimensions using None for compatibility
my_vector = np.array([10, 20])
print(my_vector.shape)
my_vector = my_vector[:, None]
print(my_vector.shape)
my_matrix + my_vector
```

```
(2,)
(2, 1)
array([[11, 12, 13],
       [24, 25, 26]])
```

Other ways to add a new dimension?

you can use `reshape` to achieve the same result as `np.newaxis`

More on matrix/vector operations

* is elementwise multiplication for `numpy.ndarray`

```
A = np.reshape(np.arange(1,13), (3, 4))
type(A)
```

```
numpy.ndarray
```

```
A
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

```
x = np.ones(4) * 2
```

```
x
```

```
array([2., 2., 2., 2.])
```

```
A * x
```

```
array([[ 2.,  4.,  6.,  8.],
       [10., 12., 14., 16.],
       [18., 20., 22., 24.]])
```

* is matrix multiplication for `numpy.matrix`

```
A = np.matrix(np.reshape(np.arange(1,13), (3, 4)))
print(type(A))
x = np.ones((4,1)) * 2
A*x
```

```
<class 'numpy.matrix'>
matrix([[20.],
        [52.],
        [84.]])
```

NumPy matrices are instances of the `numpy.matrix` class, which is a **subclass** of `numpy.ndarray`

NumPy matrices are strictly 2D and is considered somewhat outdated… Recommends using 2d arrays instead of matrices for new code

https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

More on matrix multiplication

In modern Python code (3.5+), the `@` operator is introduced (and often preferred) dealing with matrix multiplication

```
# Create two 2D NumPy arrays
A = np.array([[1, 2, 3],
              [4, 5, 6]])  # 2x3 matrix

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])  # 3x2 matrix

# Perform matrix multiplication using np.matmul()
D = np.matmul(A, B)
D
```

```
array([[ 58,  64],
       [139, 154]])
```

```
# Create two 2D NumPy arrays
A = np.array([[1, 2, 3],
              [4, 5, 6]])  # 2x3 matrix

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])  # 3x2 matrix

# Perform matrix multiplication using the @ operator
C = A @ B
```

```
C
```

```
array([[ 58,  64],
       [139, 154]])
```

Array Stack & Splitting

Common applications in:

- Data augmentation in machine learning
- Combining features in data processing
- Merging dataset
- …

Stacking in NumPy refers to joining arrays along a new axis or an existing axis.
The main functions are:

- np.vstack(): Stacks arrays vertically (row-wise)
- np.hstack(): Stacks arrays horizontally (column-wise)
- np.contatenate(): a more flexible

https://numpy.org/doc/stable/reference/generated/numpy.stack.html

Array Stack

```
import numpy as np
a = np.array([[1,2], [3,4]], dtype = 'uint')
b = np.array([[5,6], [7,8]], dtype = 'uint')
```

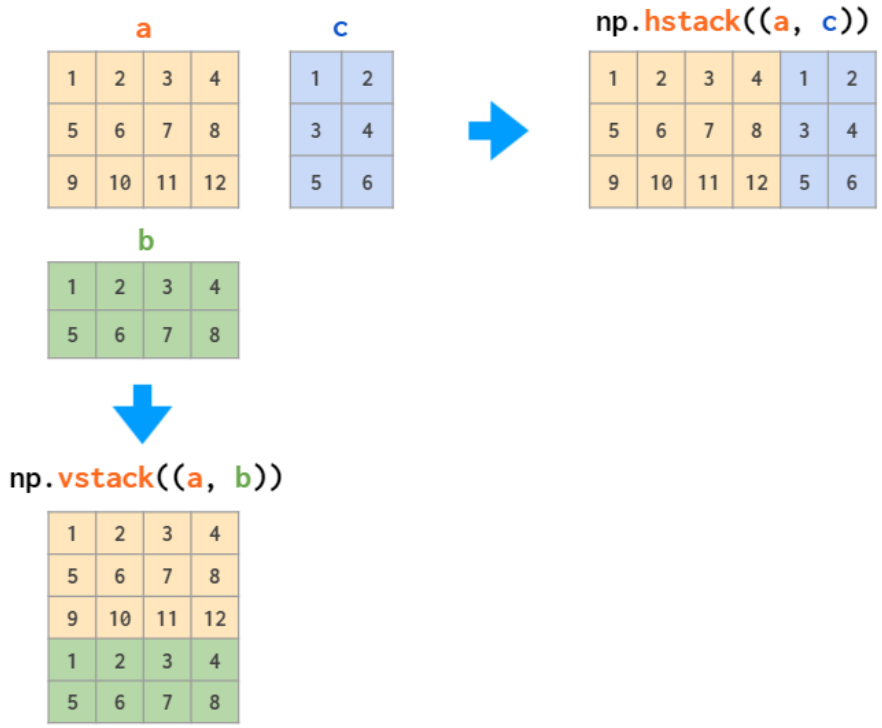
```
np.vstack((a,b))
```

```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]], dtype=uint64)
```

Always pass in a tuple, can have more than 2 NumPy arrays…

```
np.hstack((a,b))
```

```
array([[1, 2, 5, 6],
       [3, 4, 7, 8]], dtype=uint64)
```



Array Stack: `np.contatenate()`

```
np.concatenate((a1, a2, ...), axis=0, out=None, dtype=None, casting="same_kind")
```

a1, a2, …*sequence of array_like.* The arrays must have the **same shape**, except in the dimension corresponding to *axis* (the first, by default).

axis, *int, optional.* The axis along which the arrays will be joined. If axis is None, arrays are flattened before use. Default is 0.

Out, *ndarray, optional.* If provided, the destination to place the result. The shape must be correct, matching that of what concatenate would have returned if no out argument were specified.

….

```
np.concatenate((a,b),axis = 0)
```

```
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]], dtype=uint64)
```

```
np.concatenate((a,b),axis = 1)
```

```
array([[1, 2, 5, 6],
       [3, 4, 7, 8]], dtype=uint64)
```

Read more on
https://numpy.org/doc/2.0/reference/generated/numpy.concatenate.html

Array Splitting

Common applications in:

- Data partitioning for parallel computing
- Splitting dataset for cross-validation (training, testing…)
- …

`array_split`

Split an array into multiple sub-arrays of equal or near-equal size.

Does not raise an exception if an equal division cannot be made.

`hsplit`

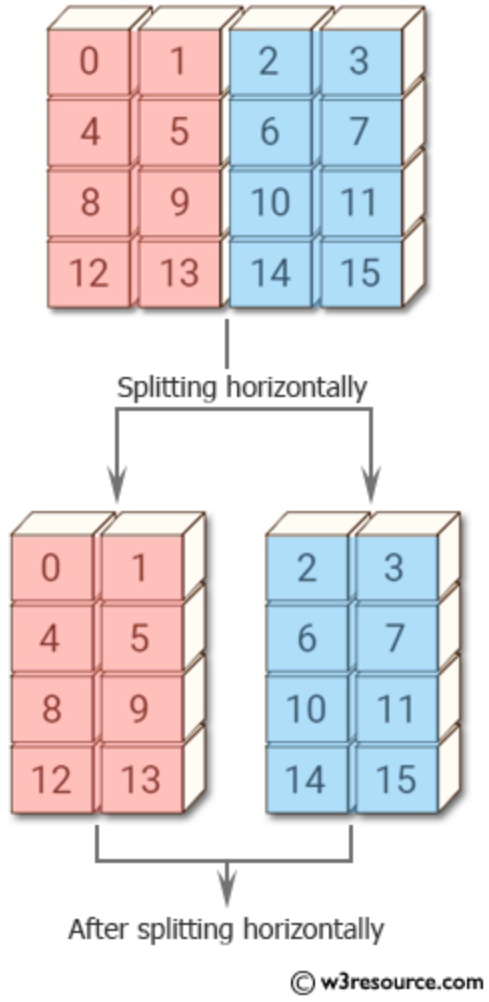
Split array into multiple sub-arrays horizontally (column-wise).

`vsplit`

Split array into multiple sub-arrays vertically (row wise).

`dsplit`

Split array into multiple sub-arrays along the 3rd axis (depth).



Read on your own in:
https://numpy.org/doc/stable/reference/generated/numpy.split.html

In-class practice

1. More on Multi-dimensional arrays operations

2. More on ufuncs

3. Statistics in NumPy

Math and ufuncs in NumPy

Universal Functions (ufuncs) in NumPy are **functions** that operate on ndarrays element-wise. They are fast and can work with broadcasting.

1. Perform **element-wise** operations on entire arrays
2. Support broadcasting, type casting, and other features.
3. Can be **faster** than using Python's built-in operators

From the documentation:

> A universal function (or ufunc for short) is a function that operates on ndarrays in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features. That is, a ufunc is a “vectorized” wrapper for a function that takes a fixed number of scalar inputs and produces a fixed number of scalar outputs.
> https://docs.scipy.org/doc/numpy/reference/ufuncs.html

So ufuncs are vectorized operations (FAST!), just like in R and MATLAB

Ufuncs practice

```python
import numpy as np

# NumPy arrays
x1 = np.array([10, 20, 30])
x2 = np.array([3, 4, 5])

# Using np.divmod() on NumPy arrays (works as a ufunc)
result_array = np.divmod(x1, x2)
print("Result with NumPy arrays:", result_array)
```

```
Result with NumPy arrays: (array([3, 5, 6]), array([1, 0, 0]))
```

`np.divmod()` is a universal function (**ufunc**) in NumPy, which means it can operate **element-wise** on NumPy arrays. This allows for **efficient vectorized operations** on array data.

which is a key feature of NumPy's **performance** benefits.

```python
# Regular Python lists
list1 = [10, 20, 30]
list2 = [3, 4, 5]

# Attempting to use python built-in divmod() on regular lists (will raise an error)
try:
    result_list = divmod(list1, list2)
    print("Result with lists:", result_list)
except TypeError as e:
    print("Error with lists:", str(e))
```

```
Error with lists: unsupported operand type(s) for divmod(): 'list' and 'list'
```

built-in `divmod()` is a NOT a universal function and it can NOT operate element-wise on Python list.

Ways to resolve this? ?

```python
result_list = list(map(divmod, list1, list2))
print("Result with built-in divmod on lists:", result_list)
```

```
Result with built-in divmod on lists: [(3, 1), (5, 0), (6, 0)]
```

Not as efficient as NumPy.

1. NumPy as numerical computing (Basics)

2. More on ufuncs

3. Statistics in NumPy

Statistics in NumPy

NumPy implements all the standard **statistics distributions/functions** you can expect

```
die_rolls = np.random.randint(1, 7, size=10)
die_rolls
```

```
array([4, 2, 4, 3, 1, 3, 2, 4, 5, 3])
```

Generate random integers from a specified range, very versatile
See `help(np.random.randint)` document.

```
errors = np.random.normal(loc=0.0, scale=1.0, size=10)
errors
```

mean std

```
array([-0.90664245, -1.20080379,  1.18109637, -0.49519272,  1.5105014 ,
        1.9763521 , -0.85794091, -1.24809495,  0.32142042,  1.30210332])
```

Examples of statistical functions provided by NumPy:

```
mean = np.mean(die_rolls)
mean
```

```
3.1
```

```
std = np.std(die_rolls)
std
```

```
1.1357816691600546
```

```
result1 = np.percentile(die_rolls, 25)
result1
```

```
2.25
```

https://numpy.org/doc/stable/reference/routines.statistics.html

Statistics in NumPy

NumPy implements all the standard **statistics distributions/functions** you can expect

```
die_rolls = np.random.randint(1, 7, size=10)
die_rolls
```

```
array([4, 2, 4, 3, 1, 3, 2, 4, 5, 3])
```

```
errors = np.random.normal(loc=0.0, scale=1.0, size=10)
errors
```

mean std

```
array([-0.90664245, -1.20080379,  1.18109637, -0.49519272,  1.5105014 ,
        1.9763521 , -0.85794091, -1.24809495,  0.32142042,  1.30210332])
```

Examples of statistical functions provided by NumPy:

```
mean = np.mean(die_rolls)
mean
```

```
3.1
```

```
std = np.std(die_rolls)
std
```

```
1.1357816691600546
```

```
result1 = np.percentile(die_rolls, 25)
result1
```

```
2.25
```

https://numpy.org/doc/stable/reference/routines.statistics.html

Random numbers in NumPy

`np.random` contains methods for generating random numbers

```
1 np.random.random((2,3))
```

A tuple representing the shape.

```
array([[ 0.61420793,  0.46363275,  0.22880783],
       [ 0.24268979,  0.13462754,  0.6026283 ]])
```

generate random numbers from a uniform distribution over [0, 1).

```
1 np.random.normal(0,1,20)
```

mean std shape(int, tuple)

```
array([ 1.31323138,  0.76807767,  1.92180038, -0.34121468,  0.72572401,
        1.0273551 , -0.78435871,  0.42732636,  1.05947171,  0.23042635,
        0.3951938 ,  0.3595342 ,  0.14710555,  0.42279814,  0.84381846,
        1.06495165, -1.51074354, -0.16419861,  2.89275956, -1.18501386])
```

```
1 np.random.uniform(0,1,(2,4))
```

(lower_boundary, upper_boundary, shape)

```
array([[ 0.08399452,  0.03934797,  0.3603464 ,  0.66361677],
       [ 0.33499095,  0.29427732,  0.14963153,  0.87892145]])
```

Lots more distributions:

https://docs.scipy.org/doc/numpy-1.17.0/reference/random/index.html

Sampling from data: `np.random.choice`

Sampling from data is a fundamental technique for data analysis.

- `np.random.choice(x,[size,replace,p])`
  - Generates a sample of `size` elements from the array `x`, drawn with (`replace=True`) or without (`replace=False`) replacement, with element probabilities given by vector `p`.

```
1 x = np.arange(1,11)
2 for i in range(5):
3     print np.random.choice(x,5,False,x/float(sum(x)))
```

```
[ 1  5 10  7  6]
[8 5 9 2 6]
[ 9  6  3  8 10]
[ 7  9 10  5  6]
[8 5 6 9 1]
```

`shuffle()` vs `permutation()`

Data randomization: randomly reordering datasets to remove potential biases

`np.random.shuffle(x)`

randomly permutes entries of `x` **in place**

return nothing (side effects)

so `x` itself is changed by this operation!

`np.random.permutation(x)`

returns a random permutation of `x`

and `x` remains unchanged.

Compare with the Python `list.sort()` and `sorted()` functions.

```
1 x = np.arange(10)
2 print x
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
1 np.random.shuffle(x)
2 print x # x is different, now.
```

```
[1 5 0 3 2 7 6 8 9 4]
```

```
1 print np.random.permutation(x)
```

```
[5 2 8 7 0 3 9 6 1 4]
```

```
1 print x # x is unchanged by permutation()
```

```
[1 5 0 3 2 7 6 8 9 4]
```

NaNs in NumPy

NaN is short for “not a number”. NaNs typically arise either because or improper mathematical operations (e.g., dividing by zero) or to represent missing data.

NumPy deals with NaNs more gracefully than MATLAB/R:

```
x = np.array([1.0,2.0,3,4,5.0])
```

```
x[4] = np.nan
```

```
x
```

```
array([ 1.,  2.,  3.,  4., nan])
```

It provides more build-in functions for dealing with NaNs

```
np.nanmean(x), np.nanmax(x), np.nanstd(x), np.nanvar(x)
```

```
(2.5, 4.0, 1.118033988749895, 1.25)
```

Compute the standard deviation along the specified axis, while **ignoring NaNs**.

Probability and statistics in SciPy

`SciPy` is a distinct Python package, part of the `numpy` ecosystem.(More on this later!)

(Almost) all the distributions you could possibly ever want:

https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions

https://docs.scipy.org/doc/scipy/reference/stats.html#multivariate-distributions

https://docs.scipy.org/doc/scipy/reference/stats.html#discrete-distributions

More statistical functions (moments, kurtosis, statistical tests):

https://docs.scipy.org/doc/scipy/reference/stats.html#statistical-functions

```
1 import scipy.stats
2 x = np.random.normal(0,1,20)
3 scipy.stats.kstest(x, 'norm')
```

```
KstestResult(statistic=0.23182037538316391, pvalue=0.1989705187485568)
```

Second argument is the name of a distribution in `scipy.stats`

Kolmogorov-Smirnov test

Other things

HW5 due this week.

Midterm next week!

Coming next:

SciPy, Matplotlib and scikit-learn (A Python ML library)