CSE 445
Lecture 3

Python Tools for Machine Learning

Datasets

- Amazon Public Data Sets
- Data.gov
- Linked Open Data
- Knowledge Bases, Encyclopedia
- Yahoo! Webscope
- Bibliography Databases
- Network/Graph Datasets
- UCI Machine Learning Repository
- Kaggle datasets
- UCR Time Series Classification/Clustering
- Time Series Data Library
- KDnuggets Dataset List
- KDD Cup Datasets

Amazon Public Data Sets

- http://aws.amazon.com/public-data-sets/
- NASA NEX: A collection of Earth science data sets maintained by NASA, including climate change projections and satellite images of the Earth's surface
- Common Crawl Corpus: A corpus of web crawl data composed of over 5 billion web pages
- 1000 Genomes Project: A detailed map of human genetic variation
- Google Books Ngrams: A data set containing Google Books n-gram corpuses
- US Census Data: US demographic data from 1980, 1990, and 2000 US Censuses
- Freebase Data Dump: A data dump of all the current facts and assertions in the Freebase system, an open database covering millions of

Data.gov

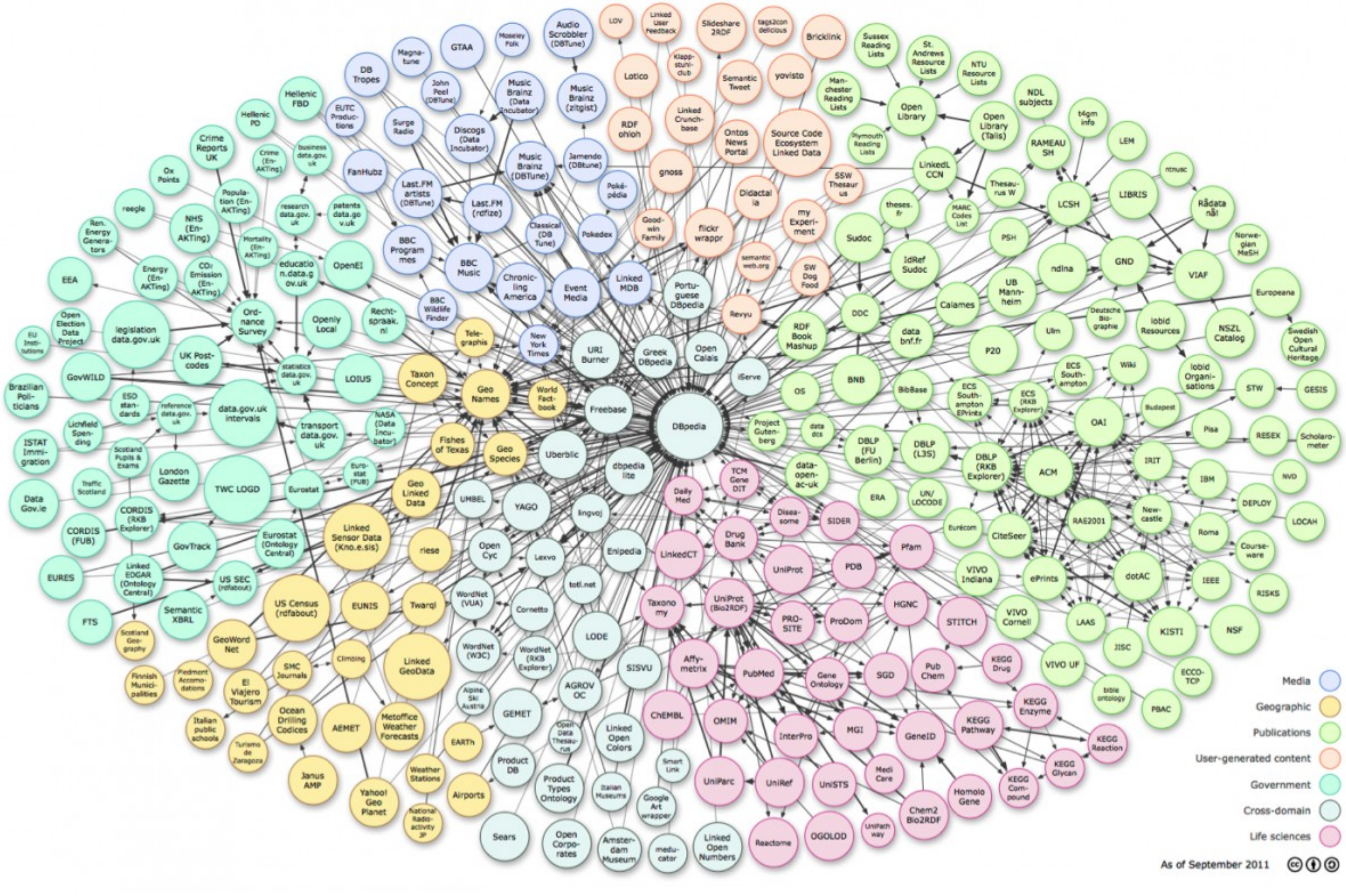
- http://www.data.gov/ (137,608 datasets)
- Consumer Complaint Database
- U.S. International Trade in Goods and Services: Monthly report that provides national trade data including imports, exports, and balance of payments for goods and services.
- DTV Reception Maps
- Climate Data Online
- Food Access Research Atlas — presents a spatial overview of food access indicators for low-income and other census tracts using different measures of supermarket...
- U.S. Hourly Precipitation Data
- Great Chile Earthquake of May 22, 1960
- Consumer Expenditure Survey
- Campus Security Data
- Farmers Markets Geographic Data: longitude and latitude, state, address, name, and zip code of Farmers Markets in the United States
- Crimes - 2001 to present (City of Chicago)

Government Data

- Government spending
- http://www.usaspending.gov/
- Campaign finance
- http://www.fec.gov/disclosure.shtml
- http://www.opensecrets.org/
- Congress voting record
- http://www.govtrack.us/ Members of Congress, Bills & Resolutions, Voting Records, Committees
- Census
- http://www.census.gov/main/www/access.html

Graph Data

- http://linkeddata.org/ (hundreds of datasets, billions of RDF triples)



Yahoo! Webscope Datasets

- Language Data
- Graph and Social Data
- Ratings and Classification Data
- Advertising and Market Data
- Competition Data
- Computing Systems Data
- Image Data

Knowledge Bases, Encyclopedia

- Wikipedia, Dbpedia
- Freebase/Google Knowledge Graph
- YAGO
- Probase
- LibraryThing

- Bibliography Databases
  - Google Scholar, Microsoft Academic Search, DBLP, arXiv.org, CiteSeer, Arnetminer
- Drug and Disease Databases
  - Drug Bank, DailyMed, OMIM, KEGG Drug
- Gene and Protein Databases
  - UniProt, Protein Data Bank, Genbank

Stanford Large Network Dataset Collection

- http://snap.stanford.edu/data/
- Social networks : online social networks, edges represent interactions between people
- Networks with ground-truth communities : ground-truth network communities in social and information networks
- Communication networks : email communication networks with edges representing communication
- Citation networks : nodes represent papers, edges represent citations
- Collaboration networks : nodes represent scientists, edges represent collaborations (co-authoring a paper)
- Web graphs : nodes represent webpages and edges are hyperlinks
- Amazon networks : nodes represent products and edges link commonly co-purchased products
- Internet networks : nodes represent computers and edges communication
- Road networks : nodes represent intersections and edges roads connecting the intersections
- …

Python tools for machine learning

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more …

Python Libraries for Data Science

NumPy:

- Introduces objects for multidimensional arrays and matrices
  - Functions to perform advanced mathematical and statistical operations on those objects

- Provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance

- Many other python libraries are built on NumPy

Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

- Collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more

- Part of SciPy Stack

- Built on NumPy

Link: https://www.scipy.org/scipylib/

Python Libraries for Data S

Pandas:

- Adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- Allows handling missing data

Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- Provides machine learning algorithms: classification, regression, clustering, model validation etc.
- Built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- A set of functionalities similar to those of MATLAB
- Line plots, scatter plots, barcharts, histograms, pie charts etc.
- Relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- Based on matplotlib
- Provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Link: https://seaborn.pydata.org/

Loading Python Libraries

In [ ]:

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Reading data using pandas

```
In [ ]: #Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None,
na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

Exploring data frames

In [3]:
```python
#List first 5 records
df.head()
```

Out[3]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

df[`salary`]

Method 2: Use the column name as an attribute:

df.salary

Note: there is an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1.

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is created we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:
df.groupby('rank')[['salary']].mean()
```

	salary
rank	
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
- by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

```
>   greater;      >= greater or equal;
<   less;         <= less or equal;
== equal;         != not equal;
```

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

```
In [ ]: #Select column salary:
        df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

```
In [ ]: #Select column salary:
        df[['rank','salary']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

In [ ]:
```python
#Select rows by their position:
df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

In [ ]:
```
#Select rows by their labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

In [ ]:
```
#Select rows by their labels:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

Data Frames: Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Missing Values

Missing values are marked as NaN

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

In [ ]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

```
In [ ]: flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R. It specifically targets statistical data visualization

To show graphs within Python notebook include inline directive:

In [ ]:
```
%matplotlib inline
```

Graphics

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Numpy

What is Numpy?

- Numpy, Scipy, and Matplotlib provide MATLAB-like functionality in python.
- Numpy Features:
  - Typed multidimentional arrays (matrices)
  - Fast numerical computations (matrix math)
  - High-level math functions

What is numpy multidimentional arrays?

- Block of memory
- How to interpret an element
- How to locate an element

Why do we need NumPy

- Python does numerical computations slowly.
- 1000 x 1000 matrix multiply
  - Python triple loop takes > 10 min.
  - Numpy takes ~0.03 seconds

NumPy Overview

1. Arrays
2. Shaping and transposition
3. Mathematical Operations
4. Indexing and slicing
5. Broadcasting

Arrays

Structured lists of numbers.

- Vectors
- Matrices
- Images
- Tensors
- ConvNets

Arrays

Structured lists of numbers.

- **Vectors**
- **Matrices**
- Images
- Tensors
- ConvNets

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Arrays

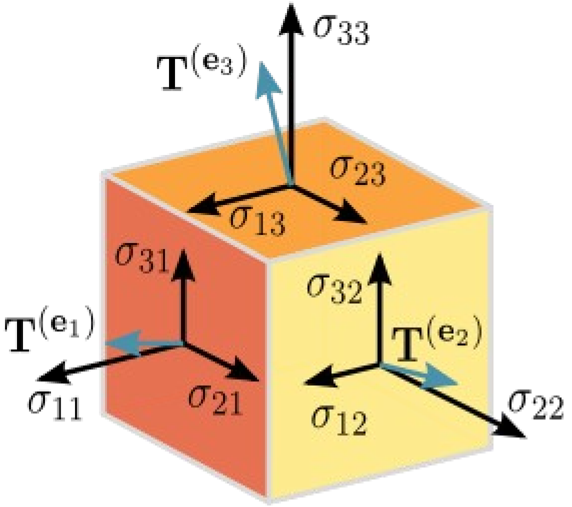
Structured lists of numbers.

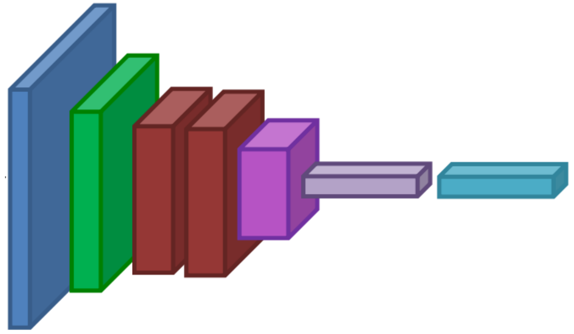
- Vectors
- Matrices
- **Images**
- Tensors
- ConvNets

Arrays

Structured lists of numbers.

- Vectors
- Matrices
- Images
- **Tensors**
- **ConvNets**





Arrays

Structured lists of numbers.

- Vectors
- Matrices
- Images
- Tensors
- ConvNets

Arrays, Basic Properties

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
Print(a.ndim, a.shape, a.dtype)
```

1. Arrays can have any number of dimensions, including zero (a scalar).
2. Arrays are typed: np.uint8, np.int64, np.float32, np.float64
3. Arrays are dense. Each element of the array exists and has the same type.

Ndarray & dtype

- A one-dimensional ndarray is a line of data; this would be a vector
- A two-dimensional ndarray would be a square of data, effectively a matrix
- A three-dimensional ndarray would be key book data, like a tensor
- Any number of dimensions is permitted, but most ndarray are one or two-dimensional
- dtype are similar to types in the basic Python language, but NumPy dtype resemble the data types seen in other languages too, such as C, C++, or Fortran, in that they are of fixed length
- dtype do have a hierarchy; a dtype usually has a string

Arrays, creation

- **np.ones, np.zeros**
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones_lik
- np.random.random

```
>>> np.ones((3,5),dtype=np.float32)
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]], dtype=float32)
```

```
>>> np.zeros((6,2),dtype=np.int8)
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```

Arrays, creation

- np.ones, np.zeros
- **np.arange**
- np.concatenate
- np.astype
- np.zeros_like, np.ones_like
- np.random.random

```
>>> np.arange(1334,1338)
array([1334, 1335, 1336, 1337])
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- **np.concatenate**
- np.astype
- np.zeros_like, np.ones_li
- np.random.random

```
>>> A = np.ones((2,3))
>>> B = np.zeros((4,3))
>>> np.concatenate([A,B])
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>>
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- **np.astype**
- np.zeros_like, np.on
- np.random.random

```
>>> A
array([[ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5],
       [ 4670.5,  4670.5,  4670.5]], dtype=float32)
>>> print(A.astype(np.uint16))
[[4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]]
```

Arrays, creation

- np.ones, np.zeros
- np.arange
- np.concatenate
- np.astype
- np.zeros_like, np.ones
- **np.random.random**

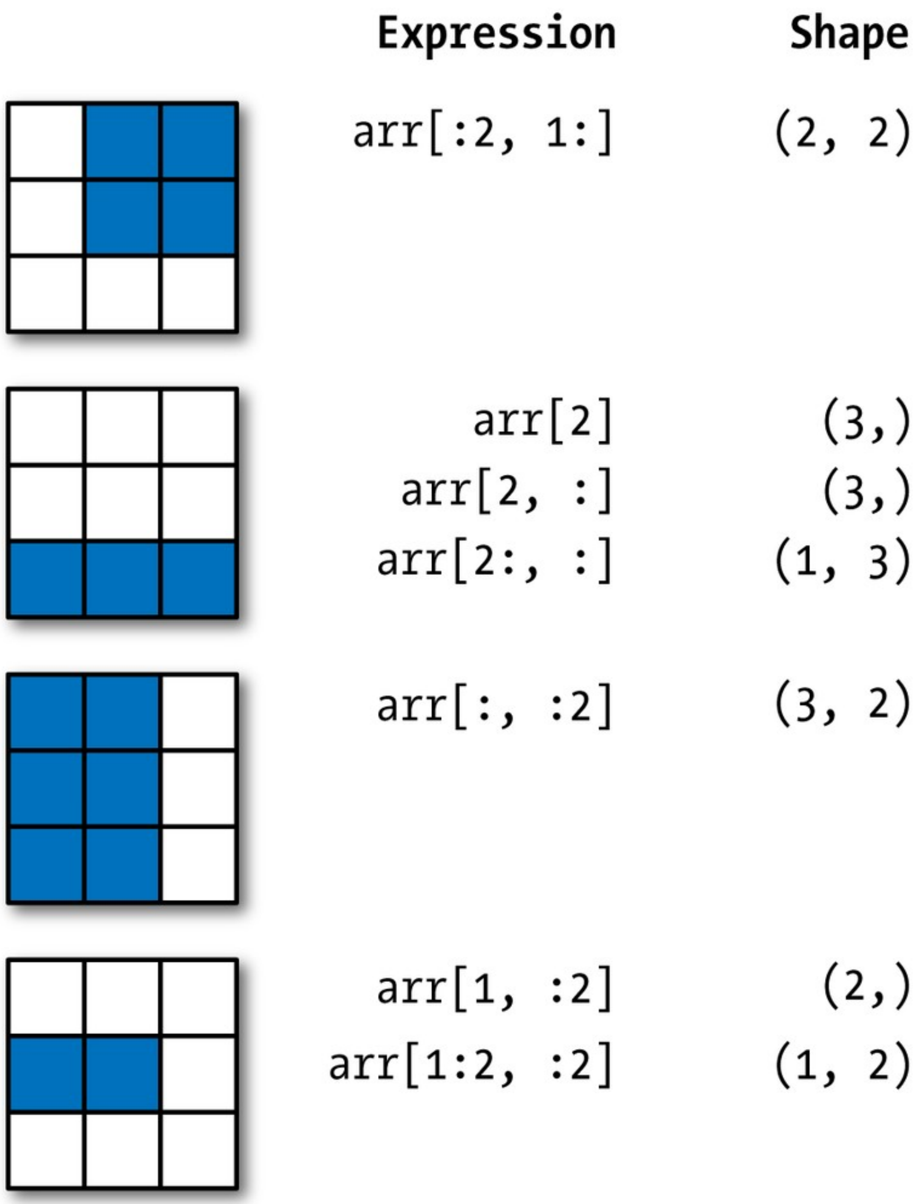
```
>>> np.random.random((10,3))
array([[ 0.61481644,  0.55453657,  0.04320502],
       [ 0.08973085,  0.25959573,  0.27566721],
       [ 0.84375899,  0.2949532 ,  0.29712833],
       [ 0.44564992,  0.37728361,  0.29471536],
       [ 0.71256698,  0.53193976,  0.63061914],
       [ 0.03738061,  0.96497761,  0.01481647],
       [ 0.09924332,  0.73128868,  0.22521644],
       [ 0.94249399,  0.72355378,  0.94034095],
       [ 0.35742243,  0.91085299,  0.15669063],
       [ 0.54259617,  0.85891392,  0.77224443]])
```

Shaping

```
a = np.array([1,2,3,4,5,6])
a = a.reshape(3,2)
a = a.reshape(2,-1)
a = a.ravel()
```

1. Total number of elements cannot change.
2. Use -1 to infer axis shape
3. Row-major by default (MATLAB is column-major)

The NumPy Array: Indexing and Slicing



Saving and loading arrays

```
np.savez('data.npz', a=a)
data = np.load('data.npz')
a = data['a']
```

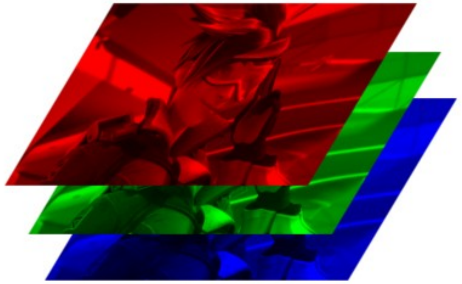
1. NPZ files can hold multiple arrays
2. np.savez_compressed similar.

Image arrays

Images are 3D arrays: width, height, and channe

Common image formats:

- height x width x RGB (band-interleaved)
- height x width (band-sequential)



Gotchas:

- Channels may also be BGR (OpenCV does this
- May be [width x height], not [height x width]

Saving and Loading Images

SciPy: `skimage.io.imread,skimage.io.imsave`

height x width x RGB

PIL / Pillow: `PIL.Image.open, Image.save`

width x height x RGB

OpenCV: `cv2.imread, cv2.imwrite`

height x width x BGR

Reading video as images: https://www.geeksforgeeks.org/extract-images-from-video-in-python/

Math, universal functions

Also called ufuncs

Element-wise

Examples:

- np.exp
- np.sqrt
- np.sin
- np.cos
- np.isnan

Math, universal functions

Also called ufuncs

Element-wise

Examples:

- np.exp
- np.sqrt
- np.sin
- np.cos
- np.isnan

```
>>> a
array([[ 1,  4],
       [ 9, 16],
       [25, 36]])
>>> np.sqrt(a)
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
```

Indexing, slices and arrays

```
I[1:-1,1:-1]    # select all but one-pixel border
I = I[:,:,::-1] # swap channel order
I[I<10] = 0   # set dark pixels to black
I[[1,3], :]   # select 2nd and 4th row
```

1. Slices are **views**. Writing to a slice overwrites the original array.
2. Can also index by a list or boolean array.