Lecture 5: Programming: NumPy, Pandas

Agenda

- Python Programming
  - NumPy, Pandas
- Later
  - Preprocessing structured data
  - Preprocessing unstructured data (time series, text, ..)

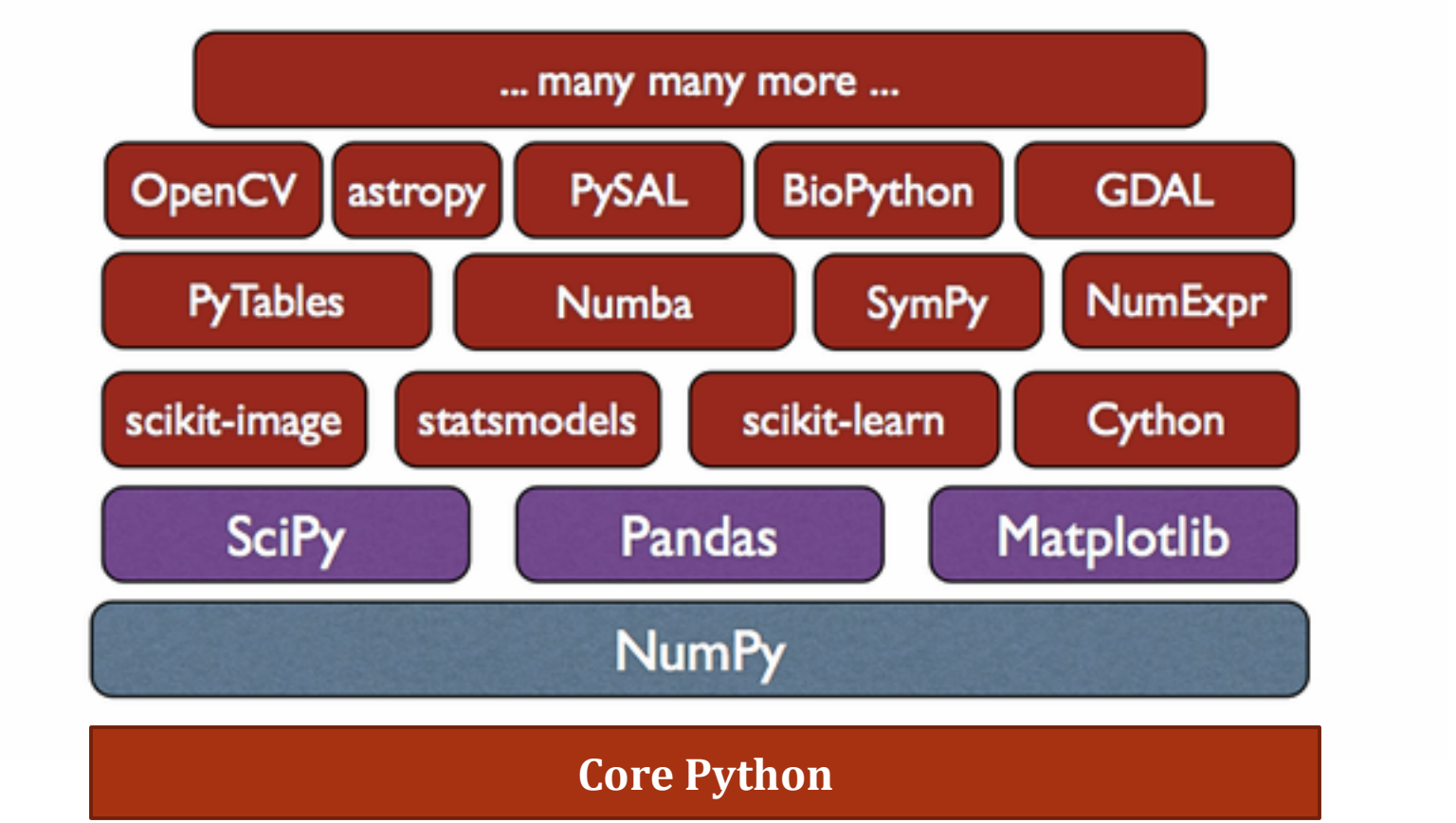
Credit

The following slides are based on:

PHY 546: Python for Scientific Computing

Instructor: Michael Zingale

Python Library Stack

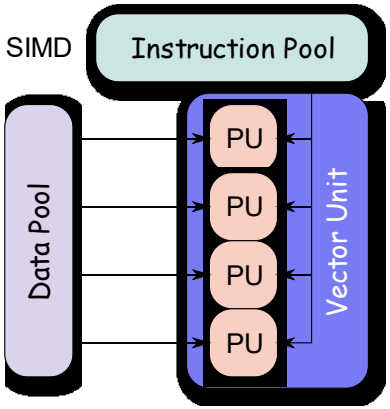


http://quasiben.github.io/PyDataEMC/#/9

Question

Which option is faster?

- Using for-loops to go over arrays
- operations on whole arrays (aka. vectorization)



Intro to NumPy: Arrays

- NumPy provides a multidimensional array
  - All elements must be the same data type
  - Many different datatypes supported

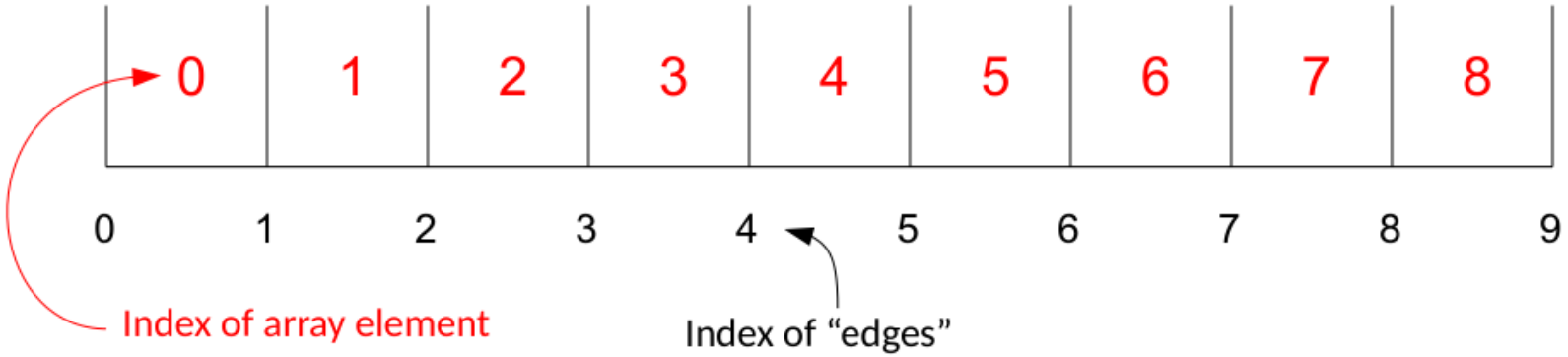
- Arithmetic operations work on arrays
- Provides MANY functions that operate on whole arrays
  - These operations are written in a compiled language, and run fast
  - Generally speaking, you want to avoid loops to get the best performance.
    - Can sometimes make code unreadable
- Lots of ways to create arrays

Intro to NumPy: Array Operations

- Arithmetic operator (+, —, /, *) work elementwise
  - `A * B` is not a matrix product, but instead multiples the corresponding elements in each array together
  - `dot(A,B)` does a dot product
- Universal functions (`sin`, `cos`, `exp`, …) work elementwise

Intro to NumPy: Array Indexing/Slicing

- Biggest source of confusion: selecting a range is best thought of as referring to the "edges" of the array locations

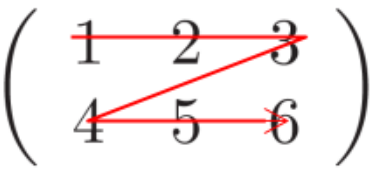


  - For the array above:
    - `A[2] = 2`
    - `A[2:3] = [2]`
    - `A[2:4] = [2 3]`
  - Note also: zero-based indexing

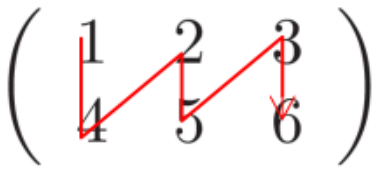
Note: this same behavior applies to Python lists and strings when slicing

Arrays

- Building block of many numerical methods
- Row vs. Column major: A(m,n)
  - First index is called the row
  - Second index is called the column
  - Multi-dimensional arrays are flattened into a one-dimensional sequence for storage
  - Row-major (C, python): rows are stored one after the other
  - Column-major (Fortran, matlab): columns are stored one after the other
- Ordering matters for:
  - Passing arrays between languages
  - Deciding which index to loop over first



Row major



Column major

Intro to NumPy: Boolean Indexing

- Many fancy ways to index arrays
- `A[A > 4] = 0`
  - Boolean indexing

Intro to NumPy: Array Views/Copies

- When "copying", need to understand if two arrays, A and B, point to:
  - the same array (including shape and data/memory space)
  - the same data/memory space (but perhaps different shapes)
  - a separate copy of the data (i.e. stored separately in memory)
- `B = A` (assignment)
  - No copy is made. A and B point to the same data in memory and share the same shape, etc.
- `B = A[:]` (view or shallow copy)
  - The shape info for A and B are stored independently, but both point to the same memory location for the data
- `B = A.copy()` (deep copy)
  - The data in B is stored completely separately in memory from A
- Copying examples…

Credit

The following slides are based on:

STAT 504 Analytics, Stephen Lee

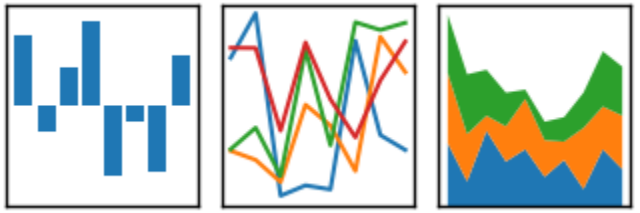
http://www.webpages.uidaho.edu/~stevel/stat504.htm

Why Pandas?

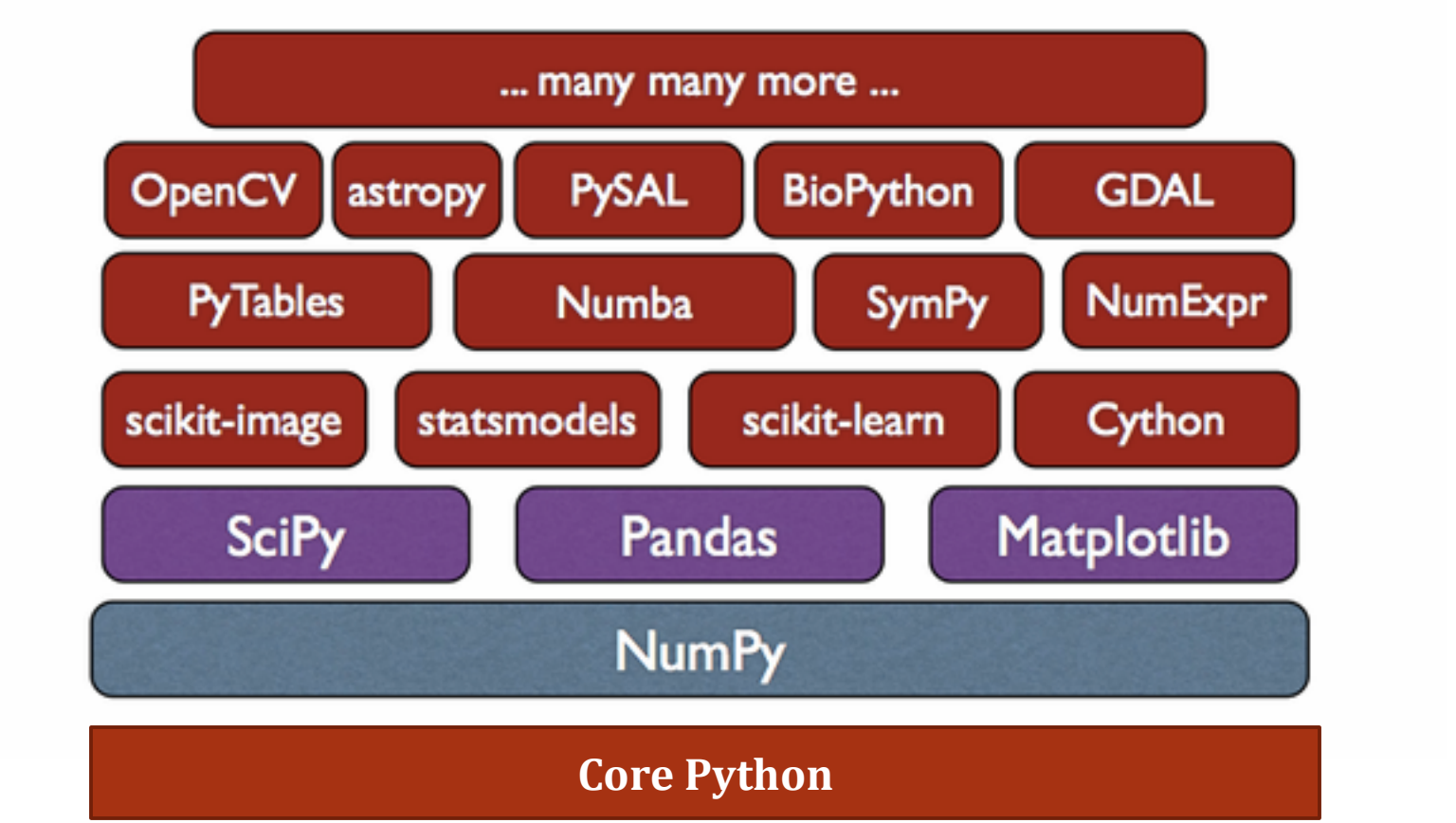
- NumPy is low-level library
- Pandas allows us to deal with data in a user-friendly; using labelled columns and indexes
- Pandas allows us to easily import data from files such as .csv files
- Pandas allows us to perform complex functions on data
- ....

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python Library Stack



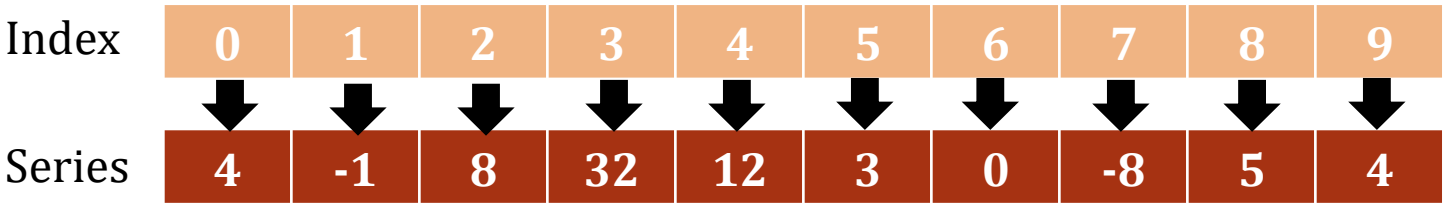
http://quasiben.github.io/PyDataEMC/#/9

Pandas Data Structures

- Series
- DataFrame

Series

- An ordered, one-dimensional array of data with an index.
- All the data in a Series is of the same data type.



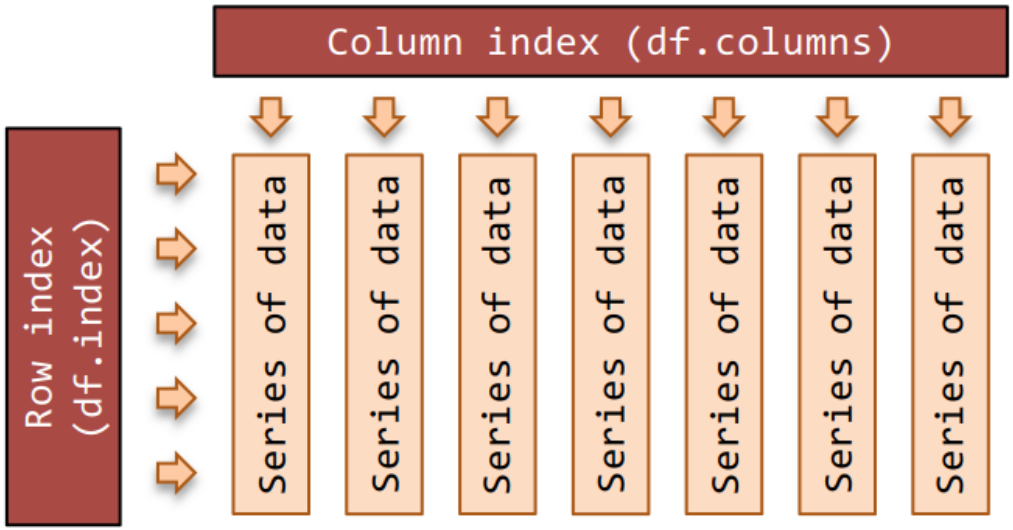
Series

- Series arithmetic is vectorized.

```
s1 = Series(range(0,4))  # -> 0, 1, 2, 3
s2 = Series(range(1,5))  # -> 1, 2, 3, 4
s3 = s1 + s2             # -> 1, 3, 5, 7
s4 = Series(['a','b'])*3 # -> 'aaa','bbb'
```

DataFrame

- The pandas DataFrame is a two-dimensional table of data with column and row indices.
- The columns are made up of pandas Series objects.



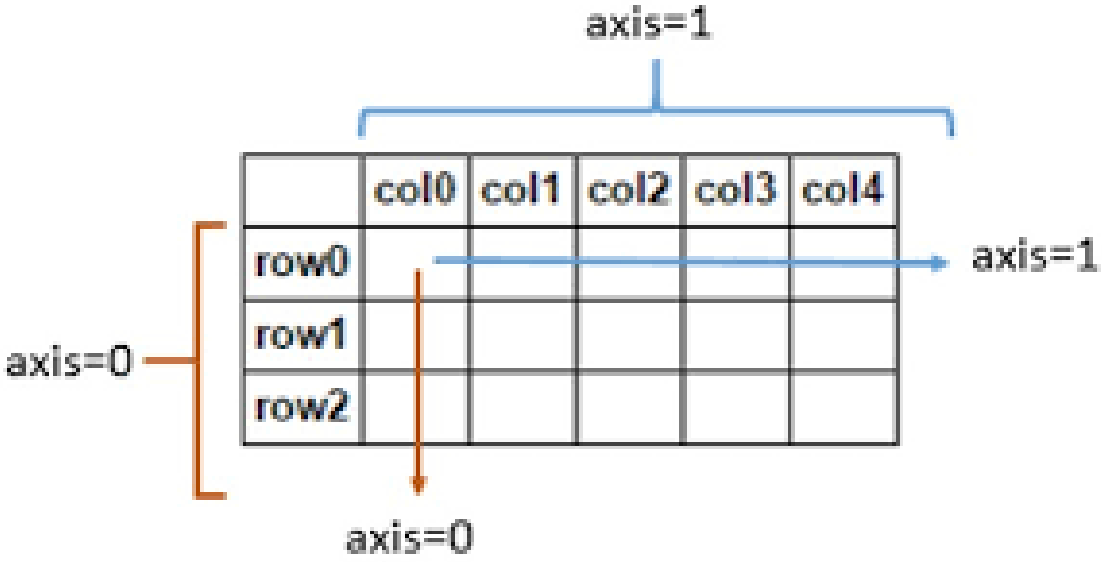
Index

- The pandas Index provides the axis labels for the Series and DataFrame objects.
- A pandas Series has one Index; and a DataFrame has two Indices.

```
# --- get Index from Series and DataFrame
idx = s.index
idx = df.columns    # the column index
idx = df.index      # the row index
```

DataFrame

- Axis 0 and axis 1 are similar to NumPy's axes



Reading and Writing Data

Load a DataFrame from a CSV file

```
df = pd.read_csv('file.csv')# often works
df = pd.read_csv('file.csv', header=0,
      index_col=0, quotechar='"',sep=':',
      na_values = ['na', '-', '.', ''])
```

Saving a DataFrame to a CSV file

```
df.to_csv('name.csv', encoding='utf-8')
```

Working with Data

Peek at the DataFrame contents

```
df.info()                # index & data types
n = 4
dfh = df.head(n)         # get first n rows
dft = df.tail(n)         # get last n rows
dfs = df.describe() # summary stats cols
top_left_corner_df = df.iloc[:5, :5]
```

Working with Data

Maths on the whole DataFrame (not a complete list)

```
df = df.abs()  # absolute values
df = df.add(o) # add df, Series or value
s = df.count() # non NA/null values
df = df.cummax()  # (cols default axis)
df = df.cummin()  # (cols default axis)
df = df.cumsum()  # (cols default axis)
df = df.cumprod() # (cols default axis)
df = df.diff() # 1st diff (col def axis)
df = df.div(o) # div by df, Series, value
df = df.dot(o) # matrix dot product
s = df.max()   # max of axis (col def)
s = df.mean()  # mean (col default axis)
s = df.median()# median (col default)
s = df.min()   # min of axis (col def)
df = df.mul(o) # mul by df Series val
s = df.sum()   # sum axis (cols default)
```

Selecting Columns

Selecting columns

```
s = df['colName'] # select col to Series
df = df[['colName']] # select col to df
df = df[['a','b']]    # select 2 or more
df = df[['c','a','b']]# change order
s = df[df.columns[0]] # select by number
df = df[df.columns[[0, 3, 4]] # by number
s = df.pop('c') # get col & drop from df
```

Python Pandas Selections and Indexing

.iloc selections - position based selection

data.iloc[<row selection], <column selection>]

Integer list of rows: [0,1,2]
Slice of rows: [4:7]
Single values: 1

Integer list of columns: [0,1,2]
Slice of columns: [4:7]
Single column selections: 1

loc selections - position based selection

data.loc[<row selection], <column selection>]

Index/Label value: 'john'
List of labels: ['john', 'sarah']
Logical/Boolean index: data['age'] == 10

Named column: 'first_name'
List of column names: ['first_name', 'age']
Slice of columns: 'first_name':'address'

For more read here