PYTHON NUMPY

Getting Python

- Download Python: https://www.python.org/
- Install Python
- Work in virtual environment
  - Recommend: Anaconda

    Anaconda install: https://docs.anaconda.com/anaconda/install/

  - Create an environment

    https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

  - Install libraries (jupyter, numpy, pandas, matplotlib, scikit-learn)

    https://docs.anaconda.com/working-with-conda/packages/install-packages/

- Jupyter Notebook: Coding in snippets

References

- Python Data Science Handbook
  - Jake VanderPlas

- https://docs.python.org/3.11/tutorial/
- http://ipython.org

Using libraries

import <library_name>

import numpy

Can rename a library with aliases

import numpy as np

Use classes and functions from library, denote library name first, then function

np.sum(variable)

Python libraries

- Numpy
- Pandas
- Scikit-learn (scipy)
- Matplotlib

Variable types

- ***numeric***: real numbers ($\mathbb{R}$)
  - double or float
- ***integer***: whole numbers ($\mathbb{Z}$)
- ***Boolean:*** True or False
- ***character***: strings
- ***raw***: binary data

Numpy

- Numerical Python

- Store and operate on data arrays

Python Lists

- A sequence of variables that may contain different types

- **Lists using range**
  L = list(range(10))
  [0,1,2,3,4,5,6,7,8,9,10]

- **Custom list**
  L = [0,1,2,3, 4,5,6,7,8,9,10]
  [0,1,2,3,4,5,6,7,8,9,10]

  L = [“a”,”b”,”c”]
  [“a”,”b”,”c”]

  L = [‘a’,’b’,’c’] -> “abc”
  “abc”

- **Iterating through lists**
  for j in L:
  print(j)

Numpy array

- Provides additional functionality
- Store and manipulate data efficiently
- Python lists can be converted into numpy array (ndarray)

  A = np.array(L)
- Can be multi-dimensional
  - Rows and columns
- Can specify data type

- Ndarray can also be created from scratch
- All zeros

  np.zeros(10,dtype=int)

  np.zeros((3,5),dtype=float)
- All ones

  np.ones((3,5),dtype=float)
- All random

  np.random.randint(10,size=(3,5))

Indexing

L = [1,3,5,7,9,11,13]

- **Examples**

L[1]	3
L[0:4]	[1,3,5,7]
L[-2]	11
L[-2:]	[11,13]
L[:-2]	[1,3,5,7,9]

- Note:
  - Indexing starts at 0
  - The last index in range is not included

Numpy array arithmetic

Operator	Equivalent function
+	np.add
-	np.subtract
*	np.multiply
/	np.divide
**	np.power
%	np.mod

Special Values

- ***None***: object has no class (its class is None)
  - Cannot be used in numpy arrays
- ***Inf***:
  - Infinity
  - Computations involving Inf returns values/Inf as expected
- ***NaN***:
  - Not a number
  - Computation involving NaN return NaN
  - Not available, a placeholder for a missing value
  - Specific functions disregard nans (e.g. nansum)

Aggregation functions in Numpy

Function Name	Nan-save Version	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute mean of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements

Pandas Objects

- Series
- DataFrames
- Index

Import statements

For remainder of presentation, assume we have following import statements

```
import pandas as pd
import numpy as np
```

Series

- One dimensional array, created from a list or array

```
data = [0.25, 0.5, 0.75, 1.0]          #create a python list
pdata = pd.Series(data)                #create a panda Series
ndata = np.array(data)                 #create a numpy array
```

- Series has 2 attributes:
  - values – list of values in the series
  - index – range of indices/specialized dictionary

Indexing

- Numpy assumes implicit indices – integers in range 0 to N (length of array)
- Series allows specialized indices (e.g. column names)
  pdata = pd.Series(data, index=['a','b','c',’d’])

- Numeric indexing
  - data[1:3] #list of 2 elements [0.5,0.75]
  - ndata[1:3] #array of 2 elements [0.5, 0.75]
  - pdata[1:3] #series of 2 elements [0.5, 0.75] with indices [b, c]

- Special indexing in pandas
  - pdata[‘b’] #numpy float
  - Similar to dictionary indexing in python
    - data = {'a':0.25,'b':0.5, 'c':0.75, 'd':1.0}
    - value = data[‘b’] #python float

DataFrames with Pandas

- Fundamental structure
- Two ways to think about DataFrames
  - Generalized Numpy array
    - Rows and columns identified by labels
  - Specialized Python dictionary
- Multiple Series concatenated into a two-dimensional array
  - area = pd.Series({'California':423967,'Texas':695662,'New York':141297,'Florida':170312,'Illinois':149995}) #area Series
  - population = pd.Series({'California':38332521,'Texas':26448193,'New York':19651127,'Florida':19552860,'Illinois':12882135}) #pop. Series
  - states = pd.DataFrame({'population':population,'area':area}) #DataFrame

Loading iris data from sklearn

```
from sklearn.datasets import load_iris
data = load_iris()                                  #data is sklearn.utils.Bunch
data
data.keys()                                         #data is similar to a dictionary
        dict_keys(['DESCR', 'feature_names', 'target', 'target_names', 'data'])
X = data.data                                       #column names are in feature_names
y = data.target                                     #class names are in target_names
df = pd.DataFrame(data.data, columns = data.feature_names)
                                                    #convert into DataFrame
```

Convert from DataFrame to Numpy

DataFrame df

```
columns = df.keys()          #column names
rownames = df.index          #row names or indices

mat = np.as array(df)        #ndarray
mat = df.values
```

Finding elements / index of element

- From iris dataset, get class of each item
  y = data.target

- Get indices of items from class o
  j = np.where(y==o)

- Get the feature weights for each of the selected items
  data.data[np.where(y==o)] or
  data.data[j] or
  mat[j]

Common mistakes in Python

- Incorrect case (most often, use lower case)
- Inconsistent indentation
- Missing/incorrect symbols
  - Especially quotes – quotes in Microsoft products are different than those used in programming