Cheat Sheet

NumPy

Basics

NumPy is a Python library for vectorised computations (similar to R), offering efficient arrays as basic strutures.

import numpy as np	load library

Arrays can only hold one data type only:

np.array([1.0, 'is', True]	gives array(['1.0','is','True'])

Initialising arrays

a=np.array([1,2,3,4])	1d array
b=np.array([[1,2],[3,4],[5,6]])	2d array
b.shape	get array shape # (3,2)
a.reshape(2, 2)	reshape a to 2x2 array
np.empty((3,2))	empty array: 3 rows, 2 cols
np.zeros((3,2))	3x2 array of zeros
np.random.random(4)	1d array of 4 random floats
np.random.random((3,2))	3x2 array of random floats
np.random.randint(1, 10, 5)	5 random integers from 1 to 10
np.random.randint(1, 10, (2,3))	5 random ints in 1 to 10 (2x3 array)
np.random.choice([1,7,9], 2, replace=True, p=[0.1,0.4,0.5])	2 random items from [1,7,9] w/o replacement & probability vector
np.random.seed(123)	seed for reproducible randomness
np.linspace(0, 10, 5)	5 evenly spaced nrs in (0, 10)

Subsetting 1d arrays

a[*from*:*to*:*step*]	basic syntax: *from* is inclusive, *to* is exclusive
a[0]	1st array element
a[-1]	last element
a[:2]	from index 0 to index 1 (!)
a[2:]	from index 2 to end
a[::2]	every 2nd element
a[::-1]	array in reverse order
a[-3:-1]	from index 3 to 1 from end

Subsetting 2d arrays

b[*rows, cols*]	basic syntax
b[*row_from*:*row_to*, *col_from*:*col_to*]	advanced syntax: *from* is inclusive, *to* is exclusive
b[0,] *or* **b[0, :]**	1st row
b[:, 0]	1st column
b[0:2,]	first two rows
b[:, 0:2]	first two cols
b[0,1]	element from 1st row, 2nd col
b[0:2, 1]	2nd col of first two rows

Conditional subsetting

a <= 3	checks condition element-wise
a[a<=3]	subset x based on condition
np.random.random(4)[x<=3]	subset array on Boolean vector from condition on x
sum(a <= 3)	count elements meeting condition (*True* evaluate to 1)
np.mean(a <= 3)	%-age of condition matches in x
np.unique(np.array(['a', 'b', 'a']))	get unique items of array

Mathematical operations

1d arrays:

np.sum(a)	sum of all elements in array x
np.median(a); np.mean(a)	mean/median of elements in x
np.var(a); np.std(a)	variance/standard deviation of x
np.min(a); np.max(a)	min/max value of x
np.argmin(a); np.argmax(a)	index of min/max value in x
np.cumsum(a)	cumulative sum at each index of x

2d arrays:

np.mean(b)	mean over all elements in array
np.mean(b, 0); np.mean(b, 1)	column-wise/row-wise means
np.around(b, 2)	round to 2 decimal places
np.nanprod(b,0); np.nanprod(b,1)	row-/col-wise product (NaN as 1)

Interoperability and comparison with Pandas

Pandas uses DataFrame as ist basic structure for data analysis.

import pandas as pd	load library
df = pd.DataFrame(b, columns = ['a', 'b'])	convert numpy array y to data frame, set col names to a and b
pd.DataFrame(a.reshape(2,2))	reshape *a* & convert to data frame
df.to_numpy()	convert data frame to numpy array

Subsetting by row/col indices

df.iloc[0,] *or* **df.iloc[0, :]**	subset 1st row
df.iloc[:, 0]	subset 1st column
df.iloc[0, 1]	1st element from 2nd column
df.iloc[0:2, 1]	2nd column of 1st two rows

Subsetting by row/col names

df.a *or* **df['a']**	subset column a
df[['a','b']]	subsect columns a and b
df.loc[:, ‘a']	subset column a
df.loc[0:1, ['a', 'b']]	1st two rows of cols a, b

Aggregation functions

df.apply(np.mean, axis=0)	column-wise aggregation
df.apply(np.mean, axis=1)	row-wise aggregation

Matrix operations on arrays

A = np.random.randint(10, size=(2, 3))	define matrix of shape 2x3
B = np.random.randint(10, size=(2, 3))	define matrix of shape 2x3
C = np.random.randint(10, size=(3,2))	define matrix of shape 3x2
np.add(A,B) *or* **A+B**	matrix addition
np.subtract(A, B) *or* **A-B**	matrix subtraction
np.multiply(A, B) *or* **A*B**	element-wise multiplication (Hadamard product)
np.divide(A, B) *or* **A / B**	
np.multiply(A, 2) *or* **A*2** ***or*** **np.dot(A, 2)**	scalar multiplication
np.matmul(A,C) *or* **A@C** ***or*** **np.dot(A,C)**	matrix multiplication
A.T *or* **np.transpose(A)**	transposition

Accessing files

np.savetxt('filename.csv', b, delimiter=",", fmt='%.3f')	save array as CSV file (floats with 3 decimal places)
np.save('filename.npy', b)	save array to binary NumPy format (*.npy*)
np.loadtxt('filename.csv', delimiter=‘,’)	load CSV file
np.loadtxt('filename.csv', delimiter=',', skiprows=1)	load CSV file omitting 1st row (e.g. header with column names)
np.load('filename.npy')	load binary NumPy file (*.npy*)
np.genfromtxt('filename.csv', delimiter=",")	load CSV with missing values set to nan

Handling images

from skimage import io	load io from *skimage* library
import matplotlib.pyplot as plt	load *pyplot* library
img = io.imread('numpy_logo.png')	load image as NumPy array
type(img)	check that image is array
img.shape	show image dimensions
plt.imshow(img)	plot image using pyplot library