Nathan Strom
HWRS 501

Numpy & Pandas Cheat Sheet

HWRS 501 – Cheat Sheet # 2

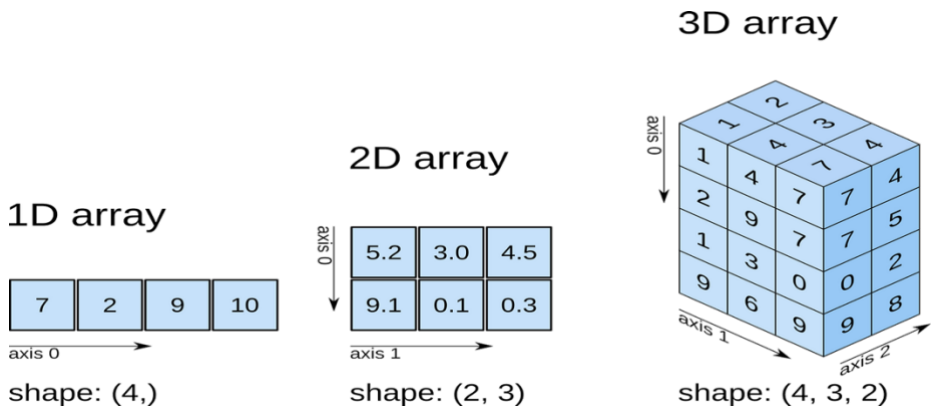
NumPy

General Purpose / Use

⇒ Provides efficient and quick functions for numerical computations.
- Especially useful for **arithmetic involving arrays and/or matrices**.
- Can perform various mathematical operations on multidimensional arrays.

NumPy Arrays

⇒ A data structure in the layout of [n: rows by m: columns] that allows the storage of numbers or other inputs.



- Especially useful for **arithmetic involving arrays and/or matrices**.
- Can perform various mathematical operations on multidimensional arrays with NumPy functions.

⇒ CAN CONTAIN
- Data of multi-dimensions: can be used to represent scalars, vectors and further.
- Data of the same type (homogenous)
- Strings (“XYZ”)
- Integers (-1, 2, 305, 43)
- Floats (4.5, 6.58, 96.543)
- Complex Numbers (3.0 + 5.4j)
- Booleans (TRUE or FALSE)
- Nested Arrays (arrays within arrays)

⇒ CANNOT CONTAIN
- Mixed data within one array (cannot be heterogenous data)

Working w/ NumPy Arrays

Counting: always starts at 0!!! The first object in a sequence: index = 0, second object: index = 1
Index of -1: returns last object

Slicing Methods

1 dimension

```
x = np.array([5, 4, 3, 2, 1, 0])

output = array([5, 4, 3, 2, 1, 0])
```

Method	Input	Output
Selection by Index	x[2]	3
Negative Indexing	x[-2]	1
Start:Stop	a) x[1:4] b) x[1:] c) x[:1] d) x[:-1] e) x[-1:]	A) [4, 3, 2] B) [4, 3, 2, 1, 0] C) [5] D) [5, 4, 3, 2, 1] E) [0]
Start:Stop:Step	x[1:4:2]	[4, 2]
Boolean Indexing	x[x<4]	[3, 2, 1, 0]

Nathan Strom
HWRS 501

Multi Dimensions

```
x3D = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

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

Method	Input	Output
Slicing	`x3D[0:2, 1:3]`	`[[2, 3],` `[5, 6]]`
Selecting by Rows & Columns [row, column]	a) `x3D[:, 1]` b) `x3D[1,: ]` c) `x3D[2, :]` d) `x3D[2, 1]`	a) `[2, 5, 8]` b) `[4, 5, 6]` c) `[7, 8, 9]` d) `8`
Step Slicing	`x3D[::2, ::2]`	`[1, 3],` `[7, 9]]`
Boolean Indexing	`x3D[x3D[:, :] > 3]`	`[4, 5, 6, 7, 8, 9]`

Creating Numpy Arrays – Various Methods

Method	Input	Output

Array method
np.array([[row#1],[row#2],[row#3]])
	`np.array([[1,2],[3,4], [5,6]])`	`[[1, 2],`
		`[3, 4],`
		`[5, 6]]`

Zero array
np.zeros(n_{rows}, $m_{columns}$)
	`np.zeros((3, 2))`	`[0., 0.],`
		`[0., 0.],`
		`[0., 0.]`

An array w/ "A Range"
np.arange(start, stop, step)
	`np.arange(10, 20, 2)`	`[10, 12,`
		`14, 16,`
		`18]`

Array to Reshape Method
x.reshape(row #, column #)
	`x = np.array([1,2,3,4,5,6,7,8,9,10,11,12])`
	or
	`x = np.arange(1, 13)`
	
	`x.reshape((2, 6))`
		`[[ 1,  2],`
		`[ 3,  4],`
		`[ 5,  6],`
		`[ 7,  8],`
		`[ 9, 10],`
		`[11, 12]]`

Nathan Strom
HWRS 501

Assignment by Indexing	`np.zeros((2,2))`
	`x[0, 0] = 1`
	`x[0, 1] = 2`
	`x[1, 0] = 3`
	`x[1, 1] = 4`
		`[1., 2.],`
		`[3., 4.]`

NumPy Functions – 6 Big Time Essentials

linespace: `np.linspace(0, 1, 10)`

- Creates an array from 0 to 1 with 10 equal spaces starting integer, ending integer, spacing)

Statistical: `np.sum(x3D), np.min(x3D), np.max(x3D), np.mean(x3D)`

- perform all these common statistical methods on the array input

Dot Product: `np.dot(x1, x2)`

- Finds the dot product of the two input arrays

Reshape array: np`.reshape(a,b )`

- Reshapes and array to a rows and b columns

Random Array with upper and lower bounds: `np.random.uniform(a, b, size = c)`

- a: lower bound, b: upper bound, c = # of columns in 1D array

Nathan Strom
HWRS 501

Pandas

General Purpose / Use

⇒ Provides an efficient way to store and manipulate data in a table (tabular) organization. Has a comprehensible user interface and allows for tasks such as selecting columns and rows based on column and row names.

Pandas vs NumPy Arrays

⇒ Data Types: Pandas data frames can have mixed data types within them. NumPy arrays cannot.
⇒ Labeling: Pandas have functionalities for labeling column and row indices/
⇒ Slicing Capabilities: Pandas allows for more splicing capabilities, to grab from data frames based upon features of interest.
⇒ Data Alignment: NumPy data is only selectable by position, Pandas data frame data can be selectable by both label-based selection and position.

Index in a dataframe

⇒ Index: list of labels that identify the rows in a PD df. The index labels are not part of the data table itself, but tools of identification and can be used for selection specification of the data of interest.

Reading a file into a Pandas Dataframe

⇒ Files in various formats can be read into as a Pandas df. Common readable files are…
  - csv
  - xlsx
  - html

⇒ Locate and note the filename, location, and data features that you want to read into as a PD df.
  - Store the file into the same folder as your python script
    - Use the function `pd.read_table("file_name", functions)`
      - Some functions that are useful for reading in files are
        - sep: identifies how data is separate. (ex: "\t" = tab separation)
        - skiprows = #: skips the amount of rows entered before reading data
        - names = ['a', 'b', 'c', 'd']: sets the names of the columns
        - index_col = ['b']: sets the b column to the index
        - parse_dates = ['b']: sets datetime functionalities in the PD df.

Nathan Strom
HWRS 501

```
data = pd.read_table('Verde_flow_wk8.txt', sep='\t', skiprows=30,
                     names =['agency_cd', 'site_no',
                             'datetime', 'flow', 'code'],index_col=['datetime'],
                             parse_dates =['datetime'])
```

Setting the Index of a Pandas Dataframe

`xdf = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6] ,'C': [7, 8, 9]}) =`

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

Multiple methods are available to set a df index…

Method 1: Set index when reading in file

⇒ Use the parameter `index_col = ['name of index']` embedded in the `pd.read_table` function

Method 2: Set index when reading in file: set_index command

⇒ `xdf = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6] , 'C': [7, 8, 9]})`

- `xdf.set_index('A')`
  - Index is now set as the 'A' column with

	B	C
A		
1	4	7
2	5	8
3	6	9

Nathan Strom
HWRS 501

Slicing Methods

```
✓ xdf['A'] ···

0    1
1    2
2    3
Name: A, dtype: int64
```

```
✓ xdf[['B']] ···

   B
0  4
1  5
2  6
```

iloc: by integer based position.

```
xdf.iloc[0:2]
✓ 0.0s

   A  B  C
0  1  4  7
1  2  5  8
```

loc: index label based

```
xdf.loc[1:2, 'B']
✓ 0.0s

1    5
2    6
```

```
xdf.loc[2,'A']
✓ 0.0s

3
```

```
xdf.loc[:, 'A']
✓ 0.0s

0    1
1    2
2    3
```

```
xdf.loc[xdf['A'] > 1]
✓ 0.0s

   A  B  C
1  2  5  8
2  3  6  9
```

Nathan Strom
HWRS 501

Pandas Functions – 6 Big Time Essentials

`df.head( )`

- Displays first 5 rows

`df.tail( )`

- Dsiplays last 5 rows
    - Can enter df.head(#) or df.tail(#)and will display # of columns

`df.describe( )`

- Displays various summary statistics for selected area of interest.

`df.sort_values(by = (name of column), ascending = (true or false)`

- Sorts values of a selected column by either ascending or descending order

`df.shape`

- Will return # rows and # columns of a dataframe

`df.reset_index( )`

- Can reset a Pandas dataframe index to the conventional pandas assigned index