# Python

leangaurav

## Numpy

leangaurav.me@gmail.com

# About Numpy

**NumPy** is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

*Source: numpy.org*



# Data Types

*ndarray*

- *ndarray* is similar to the array like types available in python (list, tuple)
- It internally uses *C* arrays to store data.

Integer Types:

- np.int8, np.int16, np.int32, np.int64
- np.uint8, np.uint16, np.uint32, np.uint64

Floating Types:

- np.float32, np.float64

# ndarray

Attributes

- flags Information about the memory layout of the array.
- shape Tuple of array dimensions.
- ndim Number of array dimensions.
- size Number of elements in the array.
- itemsize Length of one array element in bytes.
- dtype Data-type of the array's elements.
- T Transposed form of array.

ndarray is homogeneous. This is in contrast to the list and tuple types of Python



# Code

- arange(start, end, step)
- random.randint(start, end, size = <no of elements>) # default gives one no.
- linspace(start, end, count)
- zeros(shape) # shape single arg or tuple of shape
- ones(shape)

## Indexing Slicing

- Slicing works similar to normal lists
  - *array[ <row index/slice>, <column index/slice>]*
  - *array[ 1:4, [3,4] ]*
- For multidimensional slicing user the comma syntax:
  - *array[ dim1, dim2, dim3, ..... ]*
- Boolean based indexing can be used



## Any, all and NaNs

- any() checks if any True value is present, it returns True then
- all() returns True only when all elements are True
- isna() returns an ndarray of same size as input, putting True/False for each element
- Nan can't be compared with other elements and each other hence all operations in numpy on NaNs return NaN

## where function

- Used to extract indexes of element where the condition is satisfied

```
>>>where( <ndarray of bools> )
```

- Ex:

```
>>>np.where ( s %2 == 0 )
```

- Result of where can be used directly in indexing other numpy arrays



# Pandas

# What is Pandas

**pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool,
built on top of the Python programming language.

*Source: pandas.pydata.org*



# Important DataTypes

- Series
  - 1-D, like an array
  - has only one index called '*index*'

- DataFame
  - 2-D
  - has indexes for both columns and rows, called '*columns*' and '*index*' respectively

## Series creation

- pd.Series(<sequence type>)
- pd.Series(<sequence>, index=<list of corresponding index>)

- Indexes can be customized using the '*index*' option.
- Default indexes are numbers starting from *0* till *n-1*.



## Series Datatype

- Common attributes
  - shape
  - size
  - dtype
  - index
  - values
- Aggregate functions (there is one dimension, so no need of axis)
  - min, max
  - sum, mean etc.
  - unique
  - value_counts

leangaurav.me@gmail.com

# Series operations and broadcasting

- Arithmetic operations
  - +, -, *, %, / etc.
  - Series operation with a scalar broadcast it to each and every element of series
- Relational operations
  - Operators like ==, <, >, <=, >=, !=
  - These generate a corresponding series of Booleans for each element
- Logical operations
  - Like &, |, ~ etc



# Series indexing and slicing

- Series objects have only one dimension to be indexed and sliced

```
>>> <series>[ index ]

>>> <series>[ start: end: step ]
```

- Result of index is same as dtype/ type of single element
- Result of a slice is a new Series
- Boolean indexing is supported (Position where there is a True is kept)

```
>>> <series> [<series of True/False>]
```

# DataFrames

- Mostly DataFrame is created when using a function which reads data from a file format, like:
  - read_csv
  - read_excel etc.
- DataFrames can be created directly using a dictionary or a list of tuples. Each tuple denoting a row



- Common attributes
  - shape
  - size
  - dtype
  - index
  - columns
  - T
- Aggregate functions (axis =0,1 controls column or row major)
  - min, max
  - sum, mean etc.
  - Unique

- Indexing

  >>> <dataframe> [ <column name> ]

  >>> <dataframe> . <column name>

- For using second option, the column name must be a valid python identifier
- Since column names can be types other then string, hence first syntax can be used for all kinds of column names. Whether string or numeric type.



- Viewing Data

  >>> <dataframe>.head(<count>)

  >>> <dataframe>.tail(<count>)

  >>> <dataframe>.describe(include="all")

- DataFrame and Series
  - Each column in a DataFrame is a Series object.
  - Hence, all operations available on a Series are applicable to columns of a DataFrame

- Rename and in-place operations

```
>>> <dataframe>.rename(
            columns=<mapping funct/dict>,
            index = <mapping funct/dict>,
            inplace=False
    )
```

- When inplace is False, a new copy of data is returned and original DataFrame does not change
- When inplace=True, original DataFrame gets updated and a None is returned



- Indexing and Slicing: loc, iloc

```
>>> <dataframe>.loc[<rows>, <columns>]
```

loc is used to index based on row and column names

```
>>> <dataframe>.iloc[<rows>, <columns>]
```

iloc is used to index based on row and column indexes even though names might be assigned

- For slicing using loc and iloc, the : notation is used

- NA methods
  - isna Check each element is NA or not
  - fillna Fill na with values or some fill method
  - dropna Works on basis of threshold

  Usually Pandas ignores NaN values in aggregate operations unlike how it is in case of Numpy.

- Boolean Methods
  - any Anything is a true value
  - all Everything should be a true value



- Sorting/Ordering

```
>>> df.sort_values (
        by = <col/list of columns>,
        ascending=True
    )
```

- Sorts value on basis of one or more columns
- Can be used with tail and head functions to get *top-n* rows etc.

- Another option is nlargest or nsmallest

- Saving DataFrame
  - to_csv
  - to_excel etc.  Excel option requires extra operations
- DataFrames can be conveniently saved to a desired file format using any of the *'to_format'* functions.



- Aggregate operations on an entire row/column

```
>>> apply( function, axis )
```

- Element-wise operation: replace. Applies to series and/or DataFrame

```
>>> replace( dict,  regex)
```

dict can be a mapping which is applied to all columns or

nested dict {"col": { "old" : "new" }} for column wise application

if dict is regex Ex: { "col" : "$^.*?" }, regex=True should be set

- Grouping

```
>>> groupby( by=<col/list of cols>)
```

- Result is Group object
- Group objects allow all kind of aggregate functions
- Aggregate functions generate DataFrame like objects