Welcome to

Python Libraries for Machine Learning Course

reachus@cloudxlab.com

About CloudxLab

Making learning fun and for life

Videos

Quizzes

Hands-On

Projects

Case Studies

Real Life Use Cases

Automated Hands-on Assessments

Learn by doing

Automated Hands-on Assessments

Problem Statement

Hands On

Assessment

Automated Hands-on Assessments

Last Attempt Result: 3 / 87 **Last Attempt:** 1 week, 2 days ago

Getting Started With Linux Console

Please follow these steps:

1. Log into your CloudxLab Account: Open CloudxLab
2. Select the **"Credentials"** tab. You should see your login and passwords. You can copy the login and password using the icons.
3. Click on **"Web Console"** (Alternatively you could use SSH or Putty)
4. Enter your login and password. You can copy-paste from **"My Lab"**
5. If you are successfully logged in, please click on "I am Done! Please Check" button below.

Problem Statement

I am Done! Please Check

Evaluation

Edit | Jump to page | Previous | Next

Automated Hands-on Assessments

[Python Assessment]

[Jupyter Notebook]

Automated Hands-on Assessments

CLOUD x LAB

Python (Live Recordings)

Last Attempt Result:

19 / 19

Last Attempt: an hour ago

Testing Python

Define x as 10

I am Done! Please Check

✔ Good Job!

Edit | Jump to page | Previous

jupyter Untitled24

Control Panel | Logout

Menu | Trusted | Python 3

Code

In [2]:

```
x=10
```

In [3]:

```
x
```

Out[3]:

```
10
```

In [ ]:

Python Assessment

Jupyter Notebook

Course Objective

Learning Python

For

Machine Learning

&

Deep Learning

Course Instructor

Sandeep Giri

Founder CLOUD x LAB

Loves Explaining Technologies

Software Engineer

amazon INMOBI DE Shaw & Co

Worked On Large Scale Computing

Graduated from IIT Roorkee

Numpy

reachus@cloudxlab.com

What is NumPy

Stands for ***"Numeric Python"*** or ***"Numerical Python"***.

- Open Source
- Module of Python
- Provides fast mathematical functions

reachus@cloudxlab.com

What is NumPy

The complete Machine Learning eco-system.

Why use NumPy ?

- Array-oriented computing
- Efficiently implemented multi-dimensional arrays
- Designed for scientific computation
- Library of high-level mathematical functions

reachus@cloudxlab.com

Numpy - Introduction

- NumPy's main object is the homogeneous multidimensional array
- It is a table of elements
  - usually numbers
  - all of the same type
  - indexed by a tuple of positive integers
- In NumPy dimensions are called axes
- The number of axes is rank

reachus@cloudxlab.com

Numpy - Introduction

The above array has a rank of 2 since it is 2 dimensional.

Creating Numpy arrays

np.array - Creating NumPy array from Python Lists/Tuple

Numpy arrays can be created from Python lists or tuple in the following way.

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array((3, 4, 5))
>>> type(b)
<type 'numpy.ndarray'>
```


reachus@cloudxlab.com

Creating Numpy arrays

np.zeroes **-** An array with all Zeroes

To create an array with all zeroes the function np.zeroes is used

```
>>> x = np.zeros( (3,4) )
>>> x
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

Creating Numpy arrays

np.ones - An array with all Ones

To create an array with all ones the function np.ones is used.

```
>>> np.ones( (3,4), dtype=np.int16 )
array([[ 1,  1,  1,  1],
       [ 1,  1,  1,  1],
       [ 1,  1,  1,  1]])
```


reachus@cloudxlab.com

Creating Numpy arrays

np.full **-** An array with a given value

To create an array with a given shape and a given value np.full is used.

```
>>> np.full( (3,4), 0.11 )
array([[ 0.11,  0.11,  0.11,  0.11],
   [ 0.11,  0.11,  0.11,  0.11],
   [ 0.11,  0.11,  0.11,  0.11]])
```


reachus@cloudxlab.com

Creating Numpy arrays

np.arange - Creating sequence of Numbers

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )
# it accepts float arguments
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

Creating Numpy arrays

np.linspace - Creating an array with evenly distributed numbers

- Returns an array having a specific number of points
- Evenly distributed between two values
- The maximum value is included, contrary to arange

Starting Number

Ending Number

Total Number of points

```
>>> np.linspace(0, 5/3, 6)

array([0. , 0.33333333 , 0.66666667 , 1. , 1.33333333  1.66666667])
```

Creating Numpy arrays

np.random.rand - Creating an array with **random** numbers

Make a 2x3 matrix having random floats between 0 and 1:

```
>>> np.random.rand(2,3)
array([[ 0.55365951,  0.60150511,  0.36113117],
       [ 0.5388662 ,  0.06929014,  0.07908068]])
```


reachus@cloudxlab.com

Creating Numpy arrays

np.empty - Creating an empty array

To create an *uninitialised* array with a given shape. Its content is not predictable.

```
>>> np.empty((2,3))
array([[ 0.21288689,  0.20662218,  0.78018623],
       [ 0.35294004,  0.07347101,  0.54552084]])
```


reachus@cloudxlab.com

Important attributes of a NumPy object

The NumPy's array class is called ndarray. The important attributes of a ndarray object are -

ndarray.ndim

the number of axes (dimensions) of the array.

[[1., 0., 0.],

[0., 1., 2.]]

For the above array the value of ndarray.ndim is 2.

Important attributes of a NumPy object

ndarray.shape

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

For the above array the value of ndarray.shape is (2,3)

reachus@cloudxlab.com

Important attributes of a NumPy object

ndarray.size

the total number of elements of the array. This is equal to the product of the elements of shape.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

For the above array the value of ndarray.size is 6.

reachus@cloudxlab.com

Important attributes of a NumPy object

ndarray.dtype

Tells the datatype of the elements in the numpy array. All the elements in a numpy array have the same type.

```
>>> c = np.arange(1, 5)
>>> c.dtype
dtype('int64')
```


reachus@cloudxlab.com

Important attributes of a NumPy object

ndarray.itemsize

The itemsize attribute returns the size (in bytes) of each item:

```
>>> c = np.arange(1, 5)
>>> c.itemsize
8
```


reachus@cloudxlab.com

Reshaping Arrays

The function reshape is used to reshape the numpy array. The following example illustrates this.

```
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]
>>> b = a.reshape(2, 3)
>>> print(b)
[[0 1 2],
[3 4 5]]
```

Indexing and Accessing NumPy arrays

reachus@cloudxlab.com

Indexing one dimensional NumPy Arrays

0 1 2 3 4 5 6 ← Index

```
>>> a = np.array([1, 5, 3, 19, 13, 7, 3])
>>> a[3]
19
>>> a[2:5] #range
array([ 3, 19, 13])
>>> a[2::2] # How many to jump
array([ 3, 13,  3])
>>> a[::-1]  #Go reverse
array([ 3,  7, 13, 19,  3,  5,  1])
```


reachus@cloudxlab.com

Difference with regular Python arrays

1. If you assign a single value to an ndarray slice, it is copied across the whole slice :

```
>>> a = np.array([1, 2, 5, 7, 8])
>>> a[1:3] = -1
>>> a
array([ 1, -1, -1,  7,  8])
----
>>> b = [1, 2, 5, 7, 8]
>>> b[1:3] = -1
TypeError: can only assign an iterable
```


reachus@cloudxlab.com

Difference with regular Python arrays

2. ndarray slices are actually views on the same data buffer. If you modify it, it is going to modify the original ndarray as well.

```
>>> a = np.array([1, 2, 5, 7, 8])
>>> a_slice = a[1:5]
>>> a_slice[1] = 1000
>>> a
array([   1,    2, 1000, 7,    8])
# Original array was modified
```


reachus@cloudxlab.com

Important attributes of a NumPy object

3. If you want a copy of the data, you need to use the copy method as another_slice = a[2:6].copy() ,
if we modify another_slice, a remains same.

reachus@cloudxlab.com

Indexing multi dimensional NumPy arrays

Multi-dimensional arrays can be accessed as

```
>>> b[1, 2]          # row 1, col 2
>>> b[1, :]          # row 1, all columns
>>> b[:, 1]          # all rows, column 1
```

The following format is used while indexing multi-dimensional arrays

```
Array[row_start_index:row_end_index, column_start_index:
column_end_index]
```

Boolean Indexing

We can also index arrays using an ndarray of boolean values on one axis to specify the indices that we want to access.

```
>>> a = np.arange(12).reshape(3, 4)
>>> rows_on = np.array([ True, False, True])
>>> a[rows_on , : ]       # Rows 0 and 3, all columns
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11]])
```

Linear Algebra with NumPy

reachus@cloudxlab.com

Vectors

- A vector is a quantity defined by a magnitude and a direction.
- A vector can be represented by an array of numbers called scalars.

Vectors

For example, say the rocket is going up at a slight angle: it has a vertical speed of 5,000 m/s, and also a slight speed towards the East at 10 m/s, and a slight speed towards the North at 50 m/s. The rocket's velocity may be represented by the following vector:

$$\text{velocity} = \begin{pmatrix} 10 \\ 50 \\ 5000 \end{pmatrix}$$

Use of Vectors in Machine Learning

- Vectors have many purposes in Machine Learning, most notably to represent observations and predictions.
- For example, say we built a Machine Learning system to classify videos into 3 categories (good, spam, clickbait) based on what we know about them.

Use of Vectors in Machine Learning

- For each video, we would have a vector representing what we know about it, such as:

$$\text{Video} = \begin{pmatrix} 10.5 \\ 5.2 \\ 3.25 \\ 7.0 \end{pmatrix}$$

- This vector could represent a video that lasts 10.5 minutes, but only 5.2% viewers watch for more than a minute, it gets 3.25 views per day on average, and it was flagged 7 times as spam. As you can see, each axis may have a different meaning.

reachus@cloudxlab.com

Use of Vectors in Machine Learning

- Based on this vector our Machine Learning system may predict that there is an 80% probability that it is a spam video, 18% that it is clickbait, and 2% that it is a good video. This could be represented as the following vector:

$$\text{class_probabilities} = \begin{pmatrix} 0.80 \\ 0.18 \\ 0.02 \end{pmatrix} \begin{array}{l} \rightarrow \text{Spam} \\ \rightarrow \text{Clickbait} \\ \rightarrow \text{Good} \end{array}$$

reachus@cloudxlab.com

Representing Vectors in Python

- In python, a vector can be represented in many ways, the simplest being a regular python list of numbers.
  - [1,1,1,1]
- Since Machine Learning requires lots of scientific calculations, it is much better to use NumPy's ndarray, which provides a lot of convenient and optimized implementations of essential mathematical operations on vectors.
- numpy.array([1,1,1,1])

reachus@cloudxlab.com

Vectorized Operations

- Vectorized operations are far more efficient
- Than loops written in Python to do the same thing
- Let's test it

reachus@cloudxlab.com

Vectorized Operations

Matrix multiplication

1. Using for loop

```
>>> def multiply_loops(A, B):
        C = np.zeros((A.shape[0], B.shape[1]))
        for i in range(A.shape[1]):
            for j in range(B.shape[0]):
                C[i, j] = A[i, j] * B[j, i]
        return C
```

2. Using NumPy's matrix-matrix multiplication operator

```
>>> def multiply_vector(A, B):
        return A @ B
```


reachus@cloudxlab.com

Vectorized Operations

Matrix multiplication - Sample data

```
# Two randomly-generated, 100x100 matrices

>>> X = np.random.random((100, 100))
>>> Y = np.random.random((100, 100))
```


reachus@cloudxlab.com

Vectorized Operations

Matrix multiplication - Loops - timeit

```
# First, using the explicit loops:
>>> %timeit multiply_loops(X, Y)

4.23 ms ± 107 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Result - It took about 4.23 milliseconds (4.23∗10−3 seconds) to perform one matrix-matrix multiplication

Matrix multiplication - Vector - timeit

```
# Second, the NumPy multiplication:
>>> %timeit multiply_vector(X, Y)

46.6 µs ± 346 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Result - 46.6 microseconds (46.4 ∗10−6 seconds) per multiplication

Conclusion - Two orders of magnitude faster

reachus@cloudxlab.com

Basic Operations on NumPy arrays

reachus@cloudxlab.com

Addition in NumPy arrays

Addition can be performed on NumPy arrays as shown below. They apply element wise.

```
>>> a = np.array( [20, 30, 40, 50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a + b
>>> c
array([20, 31, 42, 53])
```

Subtraction in NumPy arrays

Subtraction can be performed on NumPy arrays as shown below. They apply element wise.

```
>>> a = np.array( [20, 30, 40, 50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a - b
>>> c
array([20, 29, 38, 47])
```


reachus@cloudxlab.com

Element wise product in NumPy arrays

Element wise product can be performed on NumPy arrays as shown below.

```
>>> A = np.array( [[1,1],
...               [0,1]] )
>>> B = np.array( [[2,0],
...               [3,4]] )
>>> A*B # element wise product
array([[2, 0],
       [0, 4]])
```


reachus@cloudxlab.com

Matrix Product in NumPy arrays

Matrix product can be performed on NumPy arrays as shown below.

```
>>> A = np.array( [[1,1],
...                [0,1]] )
>>> B = np.array( [[2,0],
...                [3,4]] )
>>> np.dot(A, B)                    # matrix product
array([[5, 4],
       [3, 4]])
```


reachus@cloudxlab.com

Division in NumPy arrays

Division can be performed on NumPy arrays as shown below. They apply element wise.

```
a = np.array( [20, 30, 40, 50] )
b = np.arange(1, 5)
c = a / b
c
array([ 20.          ,  15.          ,  13.33333333,  12.5
])
```


reachus@cloudxlab.com

Integer Division in NumPy arrays

Division can be performed on NumPy arrays as shown below. They apply element wise.

```
a = np.array( [20, 30, 40, 50] )
b = np.arange(1, 5)
c = a // b
c
array([20, 15, 13, 12])
```


reachus@cloudxlab.com

Modulus in NumPy arrays

Modulus operator can be applied on NumPy arrays as shown below. They apply element wise.

```
a = np.array( [20, 30, 40, 50] )
b = np.arange(1, 5)
c = a % b
c
array([0, 0, 1, 2])
```


reachus@cloudxlab.com

Exponents in NumPy arrays

We can find the exponent of each element in a NumPy array in the following way. It is applied element wise.

```
a = np.array( [20, 30, 40, 50] )
b = np.arange(1, 5)
c = a ** b
c
array([      20,      900,    64000, 6250000])
```


reachus@cloudxlab.com

Conditional Operators on NumPy arrays

Conditional operators are also applied element-wise

```
m = np.array([20, -5, 30, 40])
m < [15, 16, 35, 36]
array([False,  True,  True, False], dtype=bool)
```

```
m < 25
array([ True,  True, False, False], dtype=bool)
```

To get the elements below 25

```
m[m < 25]
array([20, -5])
```


reachus@cloudxlab.com

Broadcasting in NumPy arrays

reachus@cloudxlab.com

What is Broadcasting ?

1	2
4	5

+

0	2
3	4

→

1	4
7	9

1	2
4	5

+

5
7

→

???

reachus@cloudxlab.com

What is Broadcasting ?

In general, when NumPy expects arrays of the same shape but finds that this is not the case, it applies the so-called broadcasting rules.

Basically there are 2 rules of Broadcasting to remember.

reachus@cloudxlab.com

First rule of Broadcasting

[[[1, 3]]] + [5] → [[[6, 8]]]

Shape → (1, 1, 2) (1,) (1, 1, 2)

If the arrays do not have the same rank, then a 1 will be prepended to the smaller ranking arrays until their ranks match.

First rule of Broadcasting

```
>>> h = np.arange(5).reshape(1, 1, 5)
h
>>> array([[[0, 1, 2, 3, 4]]])
```

Let's try to add a 1D array of shape (5,) to this 3D array of shape (1,1,5), applying the first rule of broadcasting.

h + [10, 20, 30, 40, 50] # same as: h + [[[10, 20, 30, 40, 50]]]

array([[[10, 21, 32, 43, 54]]])

reachus@cloudxlab.com

Second rule of Broadcasting

Second rule of Broadcasting

On adding a 2D array of shape (2,1) to a 2D ndarray of shape (2, 3). NumPy will apply the second rule of broadcasting

```
>>> k = np.arange(6).reshape(2, 3)
>>> k
array([[0, 1, 2],
       [3, 4, 5]])

>>> k + [100, 200, 300]
array([[100, 201, 302],
       [103, 204, 305]])
```


reachus@cloudxlab.com

Mathematical and statistical functions on NumPy arrays

reachus@cloudxlab.com

Finding Mean of NumPy array elements

The ndarray object has a method mean() which finds the mean of all the elements in the array regardless of the shape of the numpy array.

```
>>> a = np.array([[-2.5, 3.1, 7], [10, 11, 12]])
>>> print("mean =", a.mean())
mean = 6.76666666667
```


reachus@cloudxlab.com

Other useful ndarray methods

Similar to mean there are other ndarray methods which can be used for various computations.

min - returns the minimum element in the ndarray
max - returns the maximum element in the ndarray
sum - returns the sum of the elements in the ndarray
prod - returns the product of the elements in the ndarray
std - returns the standard deviation of the elements in the ndarray.
var - returns the variance of the elements in the ndarray.

reachus@cloudxlab.com

Other useful ndarray methods

```
>>> a = np.array([[-2.5, 3.1, 7], [10, 11, 12]])

>>> for func in (a.min, a.max, a.sum, a.prod, a.std,
a.var):
    print(func.__name__, "=", func())

min = -2.5
max = 12.0
sum = 40.6
prod = -71610.0
std = 5.08483584352
var = 25.8555555556
```


reachus@cloudxlab.com

Summing across different axes

We can sum across different axes of a numpy array by specifying the axis parameter of the sum function.

```
>>> c=np.arange(24).reshape(2,3,4)
>>> c
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```


reachus@cloudxlab.com

Summing across different axes

```
>>> c.sum(axis=0)  # sum across matrices
  array([[12, 14, 16, 18],
         [20, 22, 24, 26],
         [28, 30, 32, 34]])
```


reachus@cloudxlab.com

Transposing Matrices

The T attribute is equivalent to calling transpose() when the rank is ≥2

```
>>> m1 = np.arange(6).reshape(2,3)
>>> m1
array([[0, 1, 2],
       [3, 4, 5]])
>>> m1.T
array([[0, 3],
       [1, 4],
       [2, 5]])
```


reachus@cloudxlab.com

Solving a system of linear scalar equations

The solve function solves a system of linear scalar equations, such as:

$$2x + 6y = 6$$
$$5x + 3y = -9$$

reachus@cloudxlab.com

Solving a system of linear scalar equations

```
>>> coeffs  = np.array([[2, 6], [5, 3]])
>>> depvars = np.array([6, -9])
>>> solution = linalg.solve(coeffs, depvars)
>>> solution
array([-3.,  2.])
```


reachus@cloudxlab.com

Solving a system of linear scalar equations

Let's check the solution.

```
>>> coeffs.dot(solution), depvars

(array([ 6., -9.]), array([ 6, -9]))
```


reachus@cloudxlab.com

Pandas

reachus@cloudxlab.com

What is Pandas?

- One of the most widely used Python libraries in Data Science after NumPy and Matplotlib
- The Pandas library Provides
  - High-performance
  - Easy-to-use data structures and
  - Data analysis tools

reachus@cloudxlab.com

Pandas - DataFrame

- The main data structure is the ***DataFrame***
- In memory 2D table
  - Like Spreadsheet with column names and row label

reachus@cloudxlab.com

Pandas - Data Analysis

- Many features available in Excel are available programmatically like
  - Creating pivot tables
  - Computing columns based on other columns
  - Plotting graphs

reachus@cloudxlab.com

Pandas - Data Structures

- ***Series objects***
  - 1D array, similar to a column in a spreadsheet
- ***DataFrame objects***
  - 2D table, similar to a spreadsheet
- ***Panel objects***
  - Dictionary of DataFrames

reachus@cloudxlab.com

Pandas - Series Objects

Creating a Series

```
>>> import pandas as pd
>>> s = pd.Series([2,-1,3,5])
```

Output -

```
0    2
1   -1
2    3
3    5
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Pass as parameters to NumPy functions

```
>>> import numpy as np
>>> np.square(s)
```

Output -

```
0     4
1     1
2     9
3    25
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Arithmetic operation on the series

```
>>> s + [1000,2000,3000,4000]
```

Output -

```
0    1002
1    1999
2    3003
3    4005
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Broadcasting

```
>>> s + 1000
```

Output -

```
0    1002
1     999
2    1003
3    1005
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Binary and conditional operations

```
>>> s < 0
```

Output -

```
0    False
1     True
2    False
3    False
dtype: bool
```


reachus@cloudxlab.com

Pandas - Series Objects

Index labels - Integer location

```
>>> s2 = pd.Series([68, 83, 112, 68])
>>> print(s2)
```

Output -

```
0     68
1     83
2    112
3     68
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Index labels - Set Manually

```
>>> s2 = pd.Series([68, 83, 112, 68],    index=["alice",
"bob", "charles", "darwin"])
>>> print(s2)
```

Output -

```
alice       68
bob         83
charles    112
darwin      68
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Access the items in series

- By specifying integer location

```
>>> s2[1]
```

- By specifying label

```
>>> s2["bob"]
```


reachus@cloudxlab.com

Pandas - Series Objects

Access the items in series - Recommendations

- Use the ***loc*** attribute when accessing by label

```
>>> s2.loc["bob"]
```

- Use ***iloc*** attribute when accessing by integer location

```
>>> s2.iloc[1]
```


reachus@cloudxlab.com

Pandas - Series Objects

Init from Python *dict*

```
>>> weights = {"alice": 68, "bob": 83, "colin": 86,
"darwin": 68}
>>> s3 = pd.Series(weights)
>>> print(s3)
```

Output -

```
alice    68
bob      83
colin    86
darwin   68
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Control the elements to include and specify their order

```
>>> s4 = pd.Series(weights, index = ["colin", "alice"])
>>> print(s4)
```

Output -

```
colin    86
alice    68
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Automatic alignment

- When an operation involves multiple Series objects
- Pandas automatically aligns items by matching index labels

reachus@cloudxlab.com

Pandas - Series Objects

Automatic alignment - example

```
>>> print(s2+s3)
```

Output -

```
alice      136.0
bob        166.0
charles      NaN
colin        NaN
darwin     136.0
dtype: float64
```

* Note ***NaN***

reachus@cloudxlab.com

Pandas - Series Objects

Automatic alignment

Do not forget to set the right index labels, else you may get surprising results

```
>>> s5 = pd.Series([1000,1000,1000,1000])
>>> print(s2 + s5)
```

Output-

```
alice     NaN
bob       NaN
charles   NaN
darwin    NaN
0         NaN
1         NaN
```


reachus@cloudxlab.com

Pandas - Series Objects

Init with a scalar

```
>>> meaning = pd.Series(42, ["life", "universe",
"everything"])
>>> print(meaning)
```

Output-

```
life        42
universe    42
everything  42
dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Series name - A Series can have a name

```
>>> s6 = pd.Series([83, 68], index=["bob", "alice"],
name="weights")
>>> print(s6)
```

* Here series name is ***weights***

Output-

```
bob      83
alice    68
Name: weights, dtype: int64
```


reachus@cloudxlab.com

Pandas - Series Objects

Plotting a series

```
>>> %matplotlib inline
>>> import matplotlib.pyplot as plt
>>> temperatures =
[4.4,5.1,6.1,6.2,6.1,6.1,5.7,5.2,4.7,4.1,3.9,3.5]
>>> s7 = pd.Series(temperatures, name="Temperature")
>>> s7.plot()
>>> plt.show()
```

Pandas - DataFrame Objects

- A DataFrame object represents
  - A spreadsheet,
  - With cell values,
  - Column names
  - And row index labels

- Visualize DataFrame as dictionaries of Series

reachus@cloudxlab.com

Pandas - DataFrame Objects

Creating a DataFrame - Pass a dictionary of Series objects

```
>>> people_dict = {
       "weight": pd.Series([68, 83, 112],index=["alice",
    "bob", "charles"]),

       "birthyear": pd.Series([1984, 1985, 1992],
index=["bob", "alice", "charles"], name="year"),

       "children": pd.Series([0, 3], index=["charles",
"bob"]),

       "hobby": pd.Series(["Biking", "Dancing"],
index=["alice", "bob"]),
}
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

Creating a DataFrame

```
>>> people = pd.DataFrame(people_dict)
>>> people
```

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68
bob	1984	3	Dancing	83
charles	1992	0	NaN	112

reachus@cloudxlab.com

Pandas - DataFrame Objects

Creating a DataFrame - Important Notes

- The Series were automatically aligned based on their index
- Missing values are represented as NaN
- Series names are ignored (the name "year" was dropped)

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68
bob	1984	3	Dancing	83
charles	1992	0	NaN	112

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Access a column

```
>>> people["birthyear"]
```

Output -

```
alice      1985
bob        1984
charles    1992
Name: birthyear, dtype: int64
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Access the multiple columns

```
>>> people[["birthyear", "hobby"]]
```

Output -

	birthyear	hobby
alice	1985	Biking
bob	1984	Dancing
charles	1992	NaN

reachus@cloudxlab.com

Pandas - DataFrame Objects

Creating DataFrame - Include columns and/or rows and guarantee order

```
>>> d2 = pd.DataFrame(
        people_dict,
        columns=["birthyear", "weight", "height"],
        index=["bob", "alice", "eugene"]
    )
>>> print(d2)
```

	birthyear	weight	height
bob	1984	83	NaN
alice	1985	68	NaN
eugene	NaN	NaN	NaN

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Accessing rows

- Using loc
  - `people.loc["charles"]`
- Using iloc
  - `People.iloc[2]`

Output -

```
birthyear    1992
children        0
hobby         NaN
weight        112
Name: charles, dtype: object
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Get a slice of rows

```
>>> people.iloc[1:3]
```

Output -

	birthyear	children	hobby	weight
bob	1984	3	Dancing	83
charles	1992	0	NaN	112

Pandas - DataFrame Objects

DataFrame - Pass a boolean array

```
>>> people[np.array([True, False, True])]
```

Output -

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68
charles	1992	0	NaN	112

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Pass boolean expression

```
>>> people[people["birthyear"] < 1990]
```

Output -

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68
bob	1984	3	Dancing	83

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Adding and removing columns

```
>>> # Adds a new column "age"
>>> people["age"] = 2016 - people["birthyear"]

>>> # Adds another column "over 30"
>>> people["over 30"] = people["age"] > 30

>>> # Removes "birthyear" and "children" columns
>>> birthyears = people.pop("birthyear")
>>> del people["children"]

>>> people
```

	hobby	weight	age	over 30
alice	Biking	68	31	True
bob	Dancing	83	32	True
charles	NaN	112	24	False

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - A new column must have the same number of rows

```
>>> # alice is missing, eugene is ignored

>>> people["pets"] = pd.Series({
        "bob": 0,
        "charles": 5,
        "eugene":1
    })

>>> people
```

	hobby	weight	age	over 30	pets
alice	Biking	68	31	True	NaN
bob	Dancing	83	32	True	0
charles	NaN	112	24	False	5

Pandas - DataFrame Objects

DataFrame - Add a new column using insert method after an existing column

```
>>> people.insert(1, "height", [172, 181, 185])
>>> people
```

	hobby	height	weight	age	over 30	pets
alice	Biking	172	68	31	True	NaN
bob	Dancing	181	83	32	True	0
charles	NaN	185	112	24	False	5

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Add new columns using assign method

```
>>> (people
        .assign(body_mass_index = lambda df:df["weight"]
        / (df["height"] / 100) ** 2)
        .assign(overweight = lambda df:
df["body_mass_index"] > 25)
     )
```

	hobby	height	weight	age	over 30	pets	body_mass_index	overweight
alice	Biking	172	68	31	True	NaN	22.985398	False
bob	Dancing	181	83	32	True	0	25.335002	True
charles	NaN	185	112	24	False	5	32.724617	True

Pandas - DataFrame Objects

DataFrame - Sorting a DataFrame

- Use sort_index method
  - It sorts the rows by their index label
  - In ascending order
  - Reverse the order by passing ***ascending=False***
  - Returns a sorted copy of DataFrame

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrame - Sorting a DataFrame

```
>>> people.sort_index(ascending=False)
```

	hobby	height	weight	age	over 30	pets	body_mass_index	overweight
charles	NaN	185	112	24	False	5	32.724617	True
bob	Dancing	181	83	32	True	0	25.335002	False
alice	Biking	172	68	31	True	NaN	22.985398	False

Pandas - DataFrame Objects

DataFrame - Sorting a DataFrame - inplace argument

```
>>> people.sort_index(inplace=True)
>>> people
```

	age	body_mass_index	height	hobby	over 30	overweight	pets	weight
alice	31	22.985398	172	Biking	True	False	NaN	68
bob	32	25.335002	181	Dancing	True	True	0.0	83
charles	24	32.724617	185	NaN	False	True	5.0	112

Pandas - DataFrame Objects

DataFrame - Sorting a DataFrame - Sort By Value

```
>>> people.sort_values(by="age", inplace=True)
>>> people
```

	age	body_mass_index	height	hobby	over 30	overweight	pets	weight
charles	24	32.724617	185	NaN	False	True	5	112
alice	31	22.985398	172	Biking	True	False	NaN	68
bob	32	25.335002	181	Dancing	True	False	0	83

Pandas - DataFrame Objects

Plotting a DataFrame

```
>>> people.plot(
        kind = "line",
        x = "body_mass_index",
        y = ["height", "weight"]
    )
>>> plt.show()
```

Pandas - DataFrame Objects

DataFrames - Saving and Loading

- Pandas can save DataFrames to various backends such as
  - CSV
  - Excel (requires openpyxl library)
  - JSON
  - HTML
  - SQL database

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Saving

Let's create a new DataFrame ***my_df*** and save it in various formats

```
>>> my_df = pd.DataFrame(
     [
       ["Biking", 68.5, 1985, np.nan],
       ["Dancing", 83.1, 1984, 3]
     ],
    columns=["hobby","weight","birthyear","children"],
    index=["alice", "bob"]
     )
>>> my_df
```

	hobby	weight	birthyear	children
alice	Biking	68.5	1985	NaN
bob	Dancing	83.1	1984	3

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Saving

- Save to CSV
  - `>>> my_df.to_csv("my_df.csv")`
- Save to HTML
  - `>>> my_df.to_html("my_df.html")`
- Save to JSON
  - `>>> my_df.to_json("my_df.json")`

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - What was saved?

```
>>> for filename in ("my_df.csv", "my_df.html",
"my_df.json"):
        print("#", filename)
        with open(filename, "rt") as f:
          print(f.read())
          print()
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - What was saved?

Note that the index is saved as the first column (with no name) in a CSV file

```
# my_df.csv
,hobby,weight,birthyear,children
alice,Biking,68.5,1985,
bob,Dancing,83.1,1984,3.0
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - What was saved?

Note that the index is saved as <th> tags in HTML

```
# my_df.html
<table border="1" class="dataframe">
  <thead>
    <tr style="text-align: right;">
      <th></th>
      <th>hobby</th>
      <th>weight</th>
      <th>birthyear</th>
      <th>children</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>alice</th>
      <td>Biking</td>
      <td>68.5</td>
      <td>1985</td>
      <td>NaN</td>
    </tr>
    <tr>
      <th>bob</th>
      <td>Dancing</td>
      <td>83.1</td>
      <td>1984</td>
      <td>3</td>
    </tr>
  </tbody>
</table>
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - What was saved?

Note that the index is saved as keys in JSON

```
# my_df.json
{"hobby":{"alice":"Biking","bob":"Dancing"},"weight":{"alice":68.
5,"bob":83.1},"birthyear":{"alice":1985,"bob":1984},"children":{"al
ice":null,"bob":3.0}}
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Loading

- `read_csv` # For loading CSV files
- `read_html` # For loading HTML files
- `read_excel` # For loading Excel files

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Load CSV file

```
>>> my_df_loaded = pd.read_csv("my_df.csv", index_col=0)

>>> my_df_loaded
```

	hobby	weight	birthyear	children
alice	Biking	68.5	1985	NaN
bob	Dancing	83.1	1984	3

Pandas - DataFrame Objects

DataFrames - Overview

- When dealing with large DataFrames, it is useful to get a quick overview of its content
- Load ***housing.csv*** inside dataset directory to create a DataFrame and get a quick overview

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Overview

- Let's understand below methods
  - `head()`
  - `tail()`
  - `info()`
  - `describe()`

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Overview - head()

- The ***head*** method returns the top 5 rows

```
>>> housing = pd.read_csv("dataset/housing.csv")
>>> housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	r
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	

Pandas - DataFrame Objects

DataFrames - Overview - tail()

- The ***tail*** method returns the bottom 5 rows
- We can also pass the number of rows we want

```
>>> housing.tail(n=2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530

reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Overview - info()

- The ***info*** method prints out the summary of each column's contents

```
>>> housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```


reachus@cloudxlab.com

Pandas - DataFrame Objects

DataFrames - Overview - describe()

- The ***describe*** method gives a nice overview of the main aggregated values over each column
  - ***count:*** number of non-null (not NaN) values
  - ***mean:*** mean of non-null values
  - ***std:*** standard deviation of non-null values
  - ***min:*** minimum of non-null values
  - ***25%, 50%, 75%:*** 25th, 50th and 75th percentile of non-null values
  - ***max:*** maximum of non-null values

reachus@cloudxlab.com

Matplotlib - Overview

- Matplotlib is a Python 2D plotting library
- Produces publication quality figures in a variety of
  - Hardcopy formats and
  - Interactive environments

reachus@cloudxlab.com

Matplotlib - Overview

- Matplotlib can be used in
  - Python scripts
  - Python and IPython shell
  - Jupyter notebook
  - Web application servers
  - GUI toolkits

Matplotlib - pyplot Module

- ***matplotlib.pyplot***
  - Collection of functions that make matplotlib work like MATLAB
  - Majority of plotting commands in ***pyplot*** have MATLAB analogs with similar arguments

reachus@cloudxlab.com

Matplotlib - pyplot Module

- ***matplotlib.pyplot***
  - Collection of functions that make matplotlib work like MATLAB
  - Majority of plotting commands in ***pyplot*** have MATLAB analogs with similar arguments

reachus@cloudxlab.com

Matplotlib - pyplot Module - plot()

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3,4])
>>> plt.ylabel('some numbers')
>>> plt.show()
```

Matplotlib - pyplot Module - plot()

plot x versus y

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
>>> plt.ylabel('some numbers')
>>> plt.show()
```

Matplotlib - pyplot Module - Histogram

```
>>> import matplotlib.pyplot as plt
>>> x =
[21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,
100]
>> num_bins = 5
>> plt.hist(x, num_bins, facecolor='blue')
>> plt.show()
```

References

- NumPy
  - https://docs.scipy.org/doc/
- Pandas
  - http://pandas.pydata.org/pandas-docs/stable/
- Matplotlib
  - https://matplotlib.org/tutorials/index.html

reachus@cloudxlab.com

Questions?

https://discuss.cloudxlab.com

reachus@cloudxlab.com

Eigenvalues and Eigenvectors

Let A be a nxn matrix.

A scalar λ is called an **eigenvalue** of A, if there is a non-zero vector x such that $Ax = \lambda x$.

Such a vector x is called **eigenvector** of A corresponding to λ.

reachus@cloudxlab.com

Eigenvalues and Eigenvectors

The eig function computes the eigenvalues and eigenvectors of a square matrix:

```
>>> import numpy.linalg as linalg
>>> A = np.array([[1,2,3],[5,7,11],[21,29,31]]) # A
>>> eigenvalues, eigenvectors = linalg.eig(A)
>>> eigenvalues # λ
array([ 42.26600592,  -0.35798416,  -2.90802176])
>>> eigenvectors # x
array([[-0.08381182, -0.76283526, -0.18913107],
       [-0.3075286 ,  0.64133975, -0.6853186 ],
       [-0.94784057, -0.08225377,  0.70325518]])
```


reachus@cloudxlab.com

Eigenvalues and Eigenvectors

Since we know that $Ax = \lambda x$ where x is the eigenvector and λ is the eigenvalue.

reachus@cloudxlab.com

Vectorized Operations

Matrix multiplication - timeit

```
# First, using the explicit loops:

>>> %timeit multiply_loops(X, Y)

4.23 ms ± 107 µs per loop (mean ± std. dev. of 7 runs,
100 loops each)
```

Result - It took about 4.23 milliseconds ($4.23*10^{-3}$ seconds) to perform one matrix-matrix multiplication

reachus@cloudxlab.com

Vectorized Operations

Matrix multiplication - timeit

```
# Now, the NumPy multiplication:

%timeit multiply_vector(X, Y)
```

```
46.6 µs ± 346 ns per loop (mean ± std. dev. of 7 runs,
10000 loops each)
```

Result - 46.6 microseconds ($46.4*10^{-6}$ seconds) per multiplication

reachus@cloudxlab.com