MDM_AI-ML_Python_Lec3

December 19, 2024

Data Analytics with Python

Lecture 3 (MDM)

By Ajit Kumar (ICT Mumbai)

Dec. 19, 2024

0.1 Variable number of arguments

0.1.1 Python *args

The special syntax ***args** in function definitions is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

[79]:
```
def myFun(*argv):
    for arg in argv:
        print(arg)
```

[81]:
```
myFun('ICT','Mumbai','India')
```

```
ICT
Mumbai
India
```

[82]:
```
myFun('Welcome', 'to', 'ICT', 'Mumbai')
```

```
Welcome
to
ICT
Mumbai
```

[16]:
```
def mysum(*args):
    result = 0
    for num in args:
        result += num
    return result
###
print(mysum(1, 2, 3, 4, 5))
```

```
15
```

[12]:
```
def myFun(arg1, *argv):
    print("First argument :", arg1)
    for arg in argv:
        print("Argument *argv :", arg)

###
myFun('Hello', 'Welcome', 'to', 'ICT', 'Mumbai')
```

```
First argument : Hello
Argument *argv : Welcome
Argument *argv : to
Argument *argv : ICT
Argument *argv : Mumbai
```

[7]:
```

```

```
First argument : Hello
Argument *argv : Welcome
Argument *argv : to
Argument *argv : ICT
Argument *argv : Mumbai
```

0.1.2 Python **kwargs

- The special syntax ****kwargs** in function definitions is used to pass a variable length argument list. We use the name kwargs with the double star **.
- Arguments are collected into a dictionary within the function that allow us to access them by their keys.

[83]:
```
def myFun(**kwargs):
    for k, val in kwargs.items():
        print("%s == %s" % (k, val))

#####
myFun(s1='ICT', s2='Means', s3='Excellence')
```

```
s1 == ICT
s2 == Means
s3 == Excellence
```

[14]:
```
def fun(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

fun(1, 2, 3, a=4, b=5,c=7)
```

```
Positional arguments: (1, 2, 3)
Keyword arguments: {'a': 4, 'b': 5, 'c': 7}
```

0.2 Numpy library

- NumPy (Numerical Python) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard library for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems.
- The NumPy library contains multidimensional array and matrix data structures. It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it.
- NumPy can be used to perform a wide variety of mathematical operations on arrays.
- It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

[84]:
```
import numpy as np  ## Importing numpy and giving it a short name 'np'
```

[86]:
```
#help(np)
```

[19]:
```
np.linspace(0,10,21) ## Dividing the interval [0,10] in 20 equal parts.
```

[19]:
```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,
        5.5,  6. ,  6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. ])
```

[20]:
```
# Generating an arithmetic progression with non integer increment.
## Contrant it with the range function
np.arange(0,5,0.5)
```

[20]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

[87]:
```
## Defining vectors
v1 = np.array([1,-2.1,3.2])
v2 = np.array([-2.3,1,-1.8])
```

[88]:
```
2*v1+3*v2
```

[88]:
```
array([-4.9, -1.2,  1. ])
```

[89]:
```
np.dot(v1,v2) # dot product of two vectors
```

[89]:
```
-10.16
```

[90]:
```
np.cross(v1,v2) # Cross product
```

[90]:
```
array([ 0.58, -5.56, -3.83])
```

[25]:
```
np.linalg.norm(v1) # length of v1  of absolute value of v1
```

[25]:
```
3.956008088970497
```

```
[91]: np.sqrt(v1.dot(v1))
```

```
[91]: 3.956008088970497
```

```
[26]: ## Orthogonal Projection of vector v1 onto v2
      v1 = np.array([1,-2.1,3.2])
      v2 = np.array([-2.3,1,-1.8])
      p = np.dot(v1,v2)/np.dot(v2,v2)*v2
      print(p)
      np.dot(v2,v1-p)
```

```
[ 2.45204617 -1.06610703  1.91899265]
```

```
[26]: 0.0
```

```
[27]: ## Angle between vectors
      u = np.array([1,2,1])
      v = np.array([3,-1,-1])
      angle = np.arccos(np.dot(u,v)/(np.linalg.norm(u)*np.linalg.norm(v)))
      print(angle)
```

```
1.5707963267948966
```

```
[28]: def angle(u,v):
          angle = np.arccos(np.dot(u,v)/(np.linalg.norm(u)*np.linalg.norm(v)))
          return angle
```

```
[31]: a = np.array([1,7,0,4])
      b = np.array([-2,5,3,-2])
      angle(a,b)
```

```
[31]: 1.0760189531792306
```

Difference between sin function in math module and numpy module

```
[98]: x = np.array([1,2,3,4,5,4,3,2,np.pi])
      np.sin(x)
```

```
[98]: array([ 8.41470985e-01,  9.09297427e-01,  1.41120008e-01, -7.56802495e-01,
             -9.58924275e-01, -7.56802495e-01,  1.41120008e-01,  9.09297427e-01,
              1.22464680e-16])
```

```
[94]: import math
      math.sin(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[94], line 2
      1 import math
```

```
----> 2 math.sin(x)

TypeError: only size-1 arrays can be converted to Python scalars
```

```
[ ]:
```

0.3 Dealing with matrices

```
[101]: A = np.array([[5,7,2],[-1,3,5],[0,1,7]])
       B = np.array([[1,-2,2],[-1,3,0],[3,1,-7]])
```

```
[33]: 3*A-2*B # linear combination of matrices
```

```
[33]: array([[13, 25,  2],
             [-1,  3, 15],
             [-6,  1, 35]])
```

```
[102]: type(A)
```

```
[102]: numpy.ndarray
```

```
[103]: A.shape
```

```
[103]: (3, 3)
```

```
[104]: A.transpose()
```

```
[104]: array([[ 5, -1,  0],
              [ 7,  3,  1],
              [ 2,  5,  7]])
```

```
[105]: A.T
```

```
[105]: array([[ 5, -1,  0],
              [ 7,  3,  1],
              [ 2,  5,  7]])
```

```
[107]: np.linalg.det(A)
```

```
[107]: 126.99999999999999
```

```
[108]: A.trace()
```

```
[108]: 15
```

```
[ ]:
```

```
[41]: print(A)
      print('')
      print(B)
```

```
[[ 5  7  2]
 [-1  3  5]
 [ 0  1  7]]

[[ 1 -2  2]
 [-1  3  0]
 [ 3  1 -7]]
```

```
[40]: A*B
```

```
[40]: array([[  5, -14,   4],
             [  1,   9,   0],
             [  0,   1, -49]])
```

```
[109]: np.dot(A,B)
```

```
[109]: array([[  4,  13,  -4],
              [ 11,  16, -37],
              [ 20,  10, -49]])
```

```
[110]: A@B
```

```
[110]: array([[  4,  13,  -4],
              [ 11,  16, -37],
              [ 20,  10, -49]])
```

```
[42]: A.T ## Transpose of A
```

```
[42]: array([[ 5, -1,  0],
             [ 7,  3,  1],
             [ 2,  5,  7]])
```

```
[43]: A.transpose() ## Transpose of A
```

```
[43]: array([[ 5, -1,  0],
             [ 7,  3,  1],
             [ 2,  5,  7]])
```

```
[44]: np.dot(A,B) # Matrix multiplication
```

```
[44]: array([[  4,  13,  -4],
             [ 11,  16, -37],
             [ 20,  10, -49]])
```

```
[45]: A@B ## Matrix multiplication
```

```
[45]: array([[  4,  13,  -4],
             [ 11,  16, -37],
             [ 20,  10, -49]])
```

```
[47]: np.linalg.det(A) # Determinant of A
```

```
[47]: 126.99999999999999
```

```
[112]: A1 = np.linalg.inv(A) #  Inverse of A
```

```
[113]: A@A1
```

```
[113]: array([[ 1.00000000e+00,  8.32667268e-17,  1.66533454e-16],
              [-1.38777878e-17,  1.00000000e+00, -2.77555756e-17],
              [ 0.00000000e+00,  1.38777878e-17,  1.00000000e+00]])
```

```
[114]: ## Solving system of linear equations
       A = np.array([[5,7,2],[-1,3,5],[0,1,7]])
       b = np.array([1,2,3])
       np.linalg.solve(A,b) # solution of Ax=b
```

```
[114]: array([ 0.07086614, -0.03149606,  0.43307087])
```

```
[115]: ## Generating a 5 x 4 matrix of random integers between 0 and 100 (both␣
        ↪inclusive)
       import random
       L= [random.randint(0,101) for i in range(20)]
       M = np.reshape(L,(5,4))
       M
```

```
[115]: array([[ 59,  95,  65,  90],
              [  5,   9,  34,   0],
              [ 12,  41,   2,  12],
              [101,   1,  87,  93],
              [101,  66, 101,  18]])
```

0.3.1 Slicing of a matrix

```
[116]: m, n = M.shape ## Prints the dimension of a matrix
```

```
[117]: M
```

```
[117]: array([[ 59,  95,  65,  90],
              [  5,   9,  34,   0],
              [ 12,  41,   2,  12],
              [101,   1,  87,  93],
              [101,  66, 101,  18]])
```

```
[121]: M[3,2]
       M[3][2]
```

```
[121]: 87
```

```
[122]: M[1:4,1:3]
```

```
[122]: array([[ 9, 34],
              [41,  2],
              [ 1, 87]])
```

```
[123]: M[:,1]
```

```
[123]: array([95,  9, 41,  1, 66])
```

0.3.2 Eigenvalues and Eigenvectors

```
[125]: M =np.reshape([random.randint(-10,10) for i in range(16)], (4,4))
       A = np.dot(M.T,M)
       ev = np.linalg.eigvals(A)
```

```
[126]: ev
```

```
[126]: array([362.52250586,  12.28811069, 205.17100872, 144.01837473])
```

```
[128]: eigval, eigvec= np.linalg.eig(A)
```

```
[131]: print(eigval)
       print(eigvec)
```

```
[362.52250586  12.28811069 205.17100872 144.01837473]
[[ 0.57725182 -0.65650457 -0.33705387  0.34953794]
 [-0.66945967 -0.36344298  0.20030698  0.61612505]
 [-0.28893657  0.28762331 -0.90094422  0.14862033]
 [ 0.36758638  0.59513342  0.18593363  0.69001822]]
```

```
[135]: print(sum(eigval))
       A.trace()
```

```
723.9999999999999
```

```
[135]: 724
```

```
[137]: np.prod(eigval)
       np.linalg.det(A)
```

```
[137]: 131629728.99999985
```

```
[139]: print(np.linalg.inv(eigvec)@A@eigvec)
```

```
[[ 3.62522506e+02 -1.42108547e-14  1.42108547e-14 -5.68434189e-14]
 [ 1.41220369e-13  1.22881107e+01  3.17523785e-14 -1.15463195e-14]
 [-3.73034936e-14 -1.42108547e-14  2.05171009e+02 -2.84217094e-14]
 [-6.39488462e-14 -5.68434189e-14 -2.84217094e-14  1.44018375e+02]]
```

```
[142]: np.round(np.linalg.inv(eigvec)@A@eigvec,4)
```

```
[142]: array([[362.5225,  -0.    ,   0.    ,  -0.    ],
              [  0.    ,  12.2881,   0.    ,  -0.    ],
              [ -0.    ,  -0.    , 205.171 ,  -0.    ],
              [ -0.    ,  -0.    ,  -0.    , 144.0184]])
```

```
[ ]:
```

```
[ ]: print(sum(ev))
     print(np.trace(M))
     print(np.prod(ev))
     print(np.linalg.det(M))
```

```
[52]: np.linalg.eig(M) # Retirns both eigenvalue and eigenvectors
```

```
[52]: (array([304.31316182, 150.89610143,   1.03868427,  54.75205249]),
       array([[ 0.4058637 ,  0.30455186, -0.84967929, -0.14341525],
              [ 0.75301517, -0.18794807,  0.3775714 , -0.50505793],
              [ 0.51612202, -0.04323849,  0.08740739,  0.85094562],
              [ 0.04313782,  0.93276692,  0.35755404, -0.01549549]]))
```

```
[53]: eig_val,eig_vec = np.linalg.eig(M)
```

```
[54]: print(eig_val)
```

```
[304.31316182 150.89610143   1.03868427  54.75205249]
```

```
[55]: print(eig_vec)
```

```
[[ 0.4058637   0.30455186 -0.84967929 -0.14341525]
 [ 0.75301517 -0.18794807  0.3775714  -0.50505793]
 [ 0.51612202 -0.04323849  0.08740739  0.85094562]
 [ 0.04313782  0.93276692  0.35755404 -0.01549549]]
```

If P is the matrix of eigenvectors of M then $P^{-1}MP$ is the diagonal matrix whose diagonal entries are eigenvalues

```
[56]: print(np.linalg.inv(eig_vec)@M@eig_vec)
```

```
[[ 3.04313162e+02  1.77635684e-15 -1.77635684e-15  1.44328993e-14]
 [ 9.76996262e-15  1.50896101e+02  1.42108547e-14  1.64313008e-14]
 [-2.85917123e-14 -1.70419234e-14  1.03868427e+00  9.94516969e-15]
 [ 5.52335955e-15  4.01900735e-14  1.75415238e-14  5.47520525e+01]]
```

[57]:
```
np.round_(np.linalg.inv(eig_vec)@M@eig_vec,decimals=6)
```

[57]:
```
array([[304.313162,   0.      ,  -0.      ,   0.      ],
       [  0.      , 150.896101,   0.      ,   0.      ],
       [ -0.      ,  -0.      ,   1.038684,   0.      ],
       [  0.      ,   0.      ,   0.      ,  54.752052]])
```

0.4 Sympy

[ ]:
```
from sympy import Matrix
a = Matrix([[4, 3], [6, 3]])
L, U, _ = a.LUdecomposition()
```

[147]:
```
L@U
```

[147]:
$$\begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix}$$

[ ]:
```

```

[143]:
```
print(L)
print(U)
```

```
Matrix([[1, 0], [3/2, 1]])
Matrix([[4, 3], [0, -3/2]])
```

[ ]:
```

```

[ ]:
```
from sympy import Matrix
A = Matrix([[12, -51, 4], [6, 167, -68], [-4, 24, -41]])
Q, R = A.QRdecomposition()
```

[151]:
```
Q@R
Q
```

[151]:
$$\begin{bmatrix} \frac{6}{7} & -\frac{69}{175} & -\frac{58}{175} \\ \frac{3}{7} & \frac{158}{175} & \frac{6}{175} \\ -\frac{2}{7} & \frac{6}{35} & -\frac{33}{35} \end{bmatrix}$$

[61]:
```
print(Q)
print(R)
```

```
Matrix([[6/7, -69/175, -58/175], [3/7, 158/175, 6/175], [-2/7, 6/35, -33/35]])
Matrix([[14, 21, -14], [0, 175, -70], [0, 0, 35]])
```

[62]:
```
A == Q*R
```

[62]:
```
True
```

[64]:
```
from sympy import Matrix
A.conjugate()
```

[64]:
$$\begin{bmatrix}12 & -51 & 4\\ 6 & 167 & -68\\ -4 & 24 & -41\end{bmatrix}$$

[65]:
```
A.adjoint()
```

[65]:
$$\begin{bmatrix}12 & 6 & -4\\ -51 & 167 & 24\\ 4 & -68 & -41\end{bmatrix}$$

[66]:
```
A.adjugate()
```

[66]:
$$\begin{bmatrix}-5215 & -1995 & 2800\\ 518 & -476 & 840\\ 812 & -84 & 2310\end{bmatrix}$$

[67]:
```
A.cofactor_matrix()
```

[67]:
$$\begin{bmatrix}-5215 & 518 & 812\\ -1995 & -476 & -84\\ 2800 & 840 & 2310\end{bmatrix}$$

[68]:
```
A.cofactor(1,2)
```

[68]:
-84

[153]:
```
M = Matrix(3, 3, [1, 2, 0, 0, 3, 0, 2, -4, 2])
(P, D) = M.diagonalize()
print(P,D)
```

```
Matrix([[-1, 0, -1], [0, 0, -1], [2, 1, 2]]) Matrix([[1, 0, 0], [0, 2, 0], [0,
0, 3]])
```

[176]:
```
A = Matrix(3, 3, [1, 2, -4, 7, 3,2, 2, -4, 2])

b = Matrix(3,1,[2,4,4])
A_aug= A.row_join(b)
A_aug
```

[176]:
$$\begin{bmatrix}1 & 2 & -4 & 2\\ 7 & 3 & 2 & 4\\ 2 & -4 & 2 & 4\end{bmatrix}$$

[181]:
```
s=A_aug.rref()[0][:,3]
s
```

[181]:
$$\begin{bmatrix}\frac{14}{13}\\ -\frac{10}{13}\\ -\frac{8}{13}\end{bmatrix}$$

```
[182]: A@s
```

[182]: $$\begin{bmatrix}2\\4\\4\end{bmatrix}$$

```
[171]: S = A_aug.rref()
```

```
[171]: (Matrix([
        [1, 0, 0,  14/13],
        [0, 1, 0, -10/13],
        [0, 0, 1,  -8/13]]),
        (0, 1, 2))
```

```
[71]: P.inv() * M * P
```

[71]: $$\begin{bmatrix}1&0&0\\0&2&0\\0&0&3\end{bmatrix}$$

```
[72]: M.eigenvals()
```

```
[72]: {3: 1, 1: 1, 2: 1}
```

```
[73]: M.eigenvects()
```

```
[73]: [(1,
        1,
        [Matrix([
         [-1/2],
         [   0],
         [   1]])]),
       (2,
        1,
        [Matrix([
         [0],
         [0],
         [1]])]),
       (3,
        1,
        [Matrix([
         [-1/2],
         [-1/2],
         [   1]])])]
```

```
[74]: from sympy import Symbol
      a = Symbol('a')
      b = Symbol('b')
      A = Matrix(3,3,[a,2,b,2,1,a,2*a,b,3])
```

```
A, A.rref(), A.echelon_form()
```

```
[74]: (Matrix([
       [  a, 2, b],
       [  2, 1, a],
       [2*a, b, 3]]),
       (Matrix([
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]]),
        (0, 1, 2)),
      Matrix([
       [2,      1,                                         a],
       [0, 4 - a,                               -a**2 + 2*b],
       [0,      0, 2*a**2*b - 8*a**2 + 4*a*b - 6*a - 4*b**2 + 24]]))
```

```
[75]: A.rref()
```

```
[75]: (Matrix([
       [1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]]),
       (0, 1, 2))
```

```
[76]: A.echelon_form()
```

[76]:
$$\begin{bmatrix} 2 & 1 & a \\ 0 & 4-a & -a^2+2b \\ 0 & 0 & 2a^2b-8a^2+4ab-6a-4b^2+24 \end{bmatrix}$$

```
[77]: from sympy import GramSchmidt
      A=Matrix(2,3,[1,0,3,2,-1,4])
      L=A.nullspace()
      GramSchmidt(L)
```

```
[77]: [Matrix([
       [-3],
       [-2],
       [ 1]])]
```

```
[78]: L
```

```
[78]: [Matrix([
       [-3],
       [-2],
       [ 1]])]
```

```
[152]: M
```

```
[152]: array([[  7,  -9,   7,   2],
               [-10,   9,   6,  -5],
               [ -3,  -4,   4, -10],
               [ -3,  -7, -10,   0]])
```

```
[ ]:
```