In [1]:
```
# Name: Vinayak Sindagi
# Reg No: 24-27-30
# Programme: M.Tech. Data Science
# Assignment Number: 2
```

In [2]:
```
import numpy as np
```

Question 1

In [4]:
```
# Part a

var1 = np.arange(31)
print('var1:\n ',var1)
print('Shape of var1 is :',var1.shape)
```

```
var1:
  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
Shape of var1 is : (31,)
```

In [5]:
```
# Part b

var2 = var1.reshape(31,1) # Since 31 elements are there shape of (5,6) or (6,5) is not possible.
print('var2:\n ',var2)
print('Shape of var2 is :',var2.shape)
```

```
var2:
  [[ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]
 [12]
 [13]
 [14]
 [15]
 [16]
 [17]
 [18]
 [19]
 [20]
 [21]
 [22]
 [23]
 [24]
 [25]
 [26]
 [27]
 [28]
 [29]
 [30]]
Shape of var2 is : (31, 1)
```

In [6]:
```
# Part c

var3 = var2.reshape(1,31,1)
print('var3:\n ',var3)
print('Shape of var3 is :',var3.shape)
```

```
var3:
  [[[ 0]
  [ 1]
  [ 2]
  [ 3]
  [ 4]
  [ 5]
  [ 6]
  [ 7]
  [ 8]
  [ 9]
  [10]
  [11]
  [12]
  [13]
  [14]
  [15]
  [16]
  [17]
  [18]
  [19]
  [20]
  [21]
  [22]
  [23]
  [24]
  [25]
  [26]
  [27]
  [28]
  [29]
  [30]]]
Shape of var3 is : (1, 31, 1)
```

In [7]:

```python
# Part d

var2[1,0] = -1      #Assigning first value in the second row of var2 to -1
print('var2 after modification:\n',var2)
print('var1 after modification:\n',var1)
print('var3 after modification:\n',var3)

# We can clearley see that:
# Both var1 and var3 is changed because reshape doesn't returns copy rather it returns view (in most of the cas
```

```
var2 after modification:
 [[ 0]
 [-1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]
 [12]
 [13]
 [14]
 [15]
 [16]
 [17]
 [18]
 [19]
 [20]
 [21]
 [22]
 [23]
 [24]
 [25]
 [26]
 [27]
 [28]
 [29]
 [30]]
var1 after modification:
 [ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
var3 after modification:
 [[[ 0]
  [-1]
  [ 2]
  [ 3]
  [ 4]
  [ 5]
  [ 6]
  [ 7]
  [ 8]
  [ 9]
  [10]
  [11]
  [12]
  [13]
  [14]
  [15]
  [16]
  [17]
  [18]
  [19]
  [20]
  [21]
  [22]
  [23]
  [24]
  [25]
  [26]
  [27]
  [28]
  [29]
  [30]]]
```

In [8]:

```python
#part e

# (i) Sum var3 over its second dimension and printing the result.
sum_over_sec = np.sum(var3,axis=1)
print('Sum of var3 over its second dimension\n',sum_over_sec)

# (ii) Sum var3 over its third dimension and print the result.
sum_over_thr = np.sum(var3,axis=2)
print('\nSum of var3 over its third dimension\n',sum_over_thr)

# (iii) Sum var3 over both first and third dimensions
sum_over_first_and_third = np.sum(var3, axis=(0, 2))
print("\nSum of var3 over both first and third dimensions:\n", sum_over_first_and_third)

# If axis=0, it sums over the first dimension (depth).
```

```
# If axis=1, it sums over the second dimension (rows).
# If axis=2, it sums over the third dimension (columns).
```

```
Sum of var3 over its second dimension
 [[463]]

Sum of var3 over its third dimension
 [[ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  24 25 26 27 28 29 30]]

Sum of var3 over both first and third dimensions:
 [ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
```

In [9]:
```
#part f

# (i) Slice out the second row of var2 and print it.
sec_row_var2 = var2[1,:]
print('Second row of Var2 is:\n',sec_row_var2)

# (ii) Slice out the last column of var2 using the -1 notation and print it.
lst_col_var2 = var2[:,-1]
print('\nLast column of var2 is:\n',lst_col_var2)

# (iii) Slice out the top right 2 × 2 submatrix of var2 and print it.
# since it is not possible,  we will create the matrix of size 6x5  and name it as var4

var4 = np.arange(1,31).reshape(5,6)
print('\nVar4 is:\n',var4)
top_rgt_matrix = var4[:2,4:]
print('\nTop right 2 × 2 submatrix of var2 is:\n',top_rgt_matrix)
```

```
Second row of Var2 is:
 [-1]

Last column of var2 is:
 [ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]

Var4 is:
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]]

Top right 2 × 2 submatrix of var2 is:
 [[ 5  6]
 [11 12]]
```

Question 2

In [11]:
```
#Part a

#

arr = np.arange(10)
print('Vector Before Broadcadcasting: ',arr)

arr1 = arr+1
print('Vector after Broadcadcasting: ',arr1)
arr1.shape
```

```
Vector Before Broadcadcasting:  [0 1 2 3 4 5 6 7 8 9]
Vector after Broadcadcasting:  [ 1  2  3  4  5  6  7  8  9 10]
```

Out[11]: (10,)

In [12]:
```
# part b

# creating a 10x10 matrix A in which Aij = i + j.

vector = arr1.reshape(10,1)
matrix = vector+arr1
print('matrix A:\n',matrix)
```

```
matrix A:
 [[ 2  3  4  5  6  7  8  9 10 11]
 [ 3  4  5  6  7  8  9 10 11 12]
 [ 4  5  6  7  8  9 10 11 12 13]
 [ 5  6  7  8  9 10 11 12 13 14]
 [ 6  7  8  9 10 11 12 13 14 15]
 [ 7  8  9 10 11 12 13 14 15 16]
 [ 8  9 10 11 12 13 14 15 16 17]
 [ 9 10 11 12 13 14 15 16 17 18]
 [10 11 12 13 14 15 16 17 18 19]
 [11 12 13 14 15 16 17 18 19 20]]
```

In [13]:
```
# part c

import numpy.random as npr
data = np.exp(npr.randn ( 50 , 5 ) ) # Creating a dummy dataset of random numbers of 50 rows and 5 columns
data
```

Out[13]:
```
array([[ 1.09567059,  0.61532612,  1.64480102,  0.52365002,  0.67823808],
       [ 0.5720722 ,  1.03580241, 16.95197698,  2.14028392,  0.92290227],
       [ 0.13730182,  0.65115957,  2.7436633 ,  2.83905584,  0.47542104],
       [ 0.68772454,  2.39544225,  1.00406316,  0.91925919,  0.91726668],
       [ 0.72794099,  0.2967898 ,  0.93517477,  0.48635787,  1.89285312],
       [ 1.10644302,  0.14468773,  1.25330527,  0.44283177,  0.41464095],
       [ 1.7800284 ,  2.84736979,  0.38467995,  1.36704178,  0.34094949],
       [ 2.61624748,  3.4308155 ,  0.19983299,  0.38517491,  0.0703507 ],
       [ 1.42429955,  0.63419086,  1.23649678,  0.48944363,  0.40377184],
       [ 2.12548523,  2.11834884,  3.53315769,  4.99500664,  2.32103121],
       [ 3.31907531,  1.14598496,  0.76533178,  0.13101549,  0.4676758 ],
       [ 2.79417503,  2.74877986,  2.06258973,  0.92859811,  2.38498831],
       [ 0.17532309,  2.21232949,  0.11516023,  0.35834783,  2.64192854],
       [ 1.08985756,  0.44762929,  0.20585944,  0.6373671 ,  1.3337626 ],
       [ 2.28919578,  1.54870203,  1.54366762,  0.41776688,  0.1136352 ],
       [ 0.19938216,  0.88236585,  1.73564084,  2.14436632,  0.28717634],
       [ 7.10468761,  0.26826354,  0.26539769,  1.38716552,  0.39517049],
       [ 0.78140837,  0.4929145 ,  1.48431804,  1.23394482,  0.50551815],
       [ 8.67832083,  0.56719225,  4.98702812,  0.45948878,  0.30480355],
       [ 0.56124184,  0.14809141,  1.45576379,  1.26931816,  1.07456147],
       [ 1.37373078,  0.83575401,  2.38906855,  1.0363349 ,  0.23432429],
       [ 0.12463701,  0.38251314, 11.92977727,  0.26200119,  0.52032248],
       [ 0.81363461,  0.64343414,  1.34574862,  1.58597835,  1.52584732],
       [ 3.17590644,  0.77559054,  0.57820611,  0.27711637,  1.48342312],
       [ 1.71638116,  4.79016127,  0.45018264,  1.22527413,  0.93510019],
       [ 0.39390542,  2.19225661,  0.44962706,  0.78188623,  4.56671229],
       [ 0.60554372,  2.80318673,  0.73402606,  0.08930492,  2.10458113],
       [ 1.06568665,  0.30971815,  0.25033995,  0.3232014 ,  1.23559445],
       [ 1.77508195,  1.20903211,  0.40082837,  0.2589789 ,  0.31983878],
       [ 0.82362028,  1.30423694,  1.41564256,  0.25353538,  0.18110337],
       [ 1.19353604,  0.1584304 ,  0.35912429,  0.9997565 ,  1.34858562],
       [ 0.33137172,  0.21501404,  2.01639552,  0.36588976,  1.53874674],
       [ 0.36544269,  4.78602515,  0.14929745,  9.89799724,  2.81305252],
       [ 8.53701922,  1.04052387,  0.95690993,  0.92560468,  2.04859374],
       [ 0.43789928,  1.18405907,  0.1703702 ,  0.73267276,  0.42954095],
       [ 0.69193307,  1.68341183,  0.43866661,  0.95227324,  1.2683716 ],
       [ 1.19137388,  2.10622041,  0.58432318,  1.2259338 ,  0.81926395],
       [ 2.67839131,  1.06829574,  0.24239624,  2.53894465,  0.84338989],
       [ 0.56936655,  0.73577008,  0.77518318,  0.58268672,  0.66232339],
       [ 2.88326502,  0.37859445,  0.66500408,  1.45368858,  1.26370246],
       [ 2.30862382,  1.2645932 ,  0.33306262,  2.35232227,  0.69565473],
       [ 2.52033258,  0.49894859,  0.18116976,  1.54622356,  1.32672336],
       [ 0.89890598,  1.07299235,  0.48979155,  2.26838716,  0.1552863 ],
       [ 1.53295975,  0.19681512,  0.98397797,  1.17387806, 13.32971083],
       [ 1.77317397,  0.99149113,  0.90651428,  0.77426591,  0.74700071],
       [ 0.69307322,  3.5823352 ,  3.19040292,  0.95946419,  0.23091737],
       [ 2.61430192,  3.04298787,  0.40947274,  0.59078541,  1.05225766],
       [ 0.59032409, 12.23781585,  0.66644689,  0.68175997,  0.783514  ],
       [ 0.65397885,  2.22071626,  0.29968024,  0.22116945,  5.38169094],
       [ 0.73065735,  0.84041763,  1.91981427,  0.31827323,  0.46588299]])
```

In [14]:
```
#part e

# Calculating mean and standard deviation of each column

mean = np.mean(data,axis = 0)
std = np.std(data,axis = 0)
print('Mean of each column:',mean)
print('Standard deviation of each column: ',std)
```

```
Mean of each column: [1.68659879 1.58367056 1.60378721 1.20422147 1.36515406]
Standard deviation of each column:  [1.84544131 1.90626284 2.83931169 1.51774477 2.00343444]
```

In [15]:
```
# part f
# Standardizing the data and finding the mean and standard deviation
```

```
normalized = (data-mean)/std
print('Normalized data =\n',normalized)

mean_nor = np.mean(normalized,axis = 0)
std_nor = np.std(normalized,axis = 0)

print('\nMean of each column:',mean_nor)
print('\nStandard deviation of each column: ',std_nor)
```

```
Normalized data =
 [[-0.3202097  -0.50798055  0.01444499 -0.44840968 -0.34286921]
 [-0.603935   -0.28740431  5.40560229  0.61674563 -0.22074682]
 [-0.83952655 -0.4891828   0.40146212  1.0771471  -0.44410389]
 [-0.54126579  0.42584458 -0.21122163 -0.18775376 -0.22355979]
 [-0.51947347 -0.67508044 -0.23548399 -0.47298045  0.26339722]
 [-0.31437238 -0.75487115 -0.12343905 -0.50165859 -0.47444183]
 [ 0.05062724  0.66291972 -0.42936718  0.10727779 -0.5112244 ]
 [ 0.50375413  0.96898754 -0.49446992 -0.5396471  -0.64629186]
 [-0.14213362 -0.49808436 -0.12935897 -0.47094732 -0.47986707]
 [ 0.23782194  0.28048508  0.6795205   2.49764338  0.47711926]
 [ 0.88459953 -0.22960402 -0.29530235 -0.7071057  -0.44796987]
 [ 0.60016877  0.61120077  0.16158935 -0.1816006   0.50904299]
 [-0.81892374  0.32978607 -0.5242915  -0.55732272  0.63729287]
 [-0.32335964 -0.59595206 -0.49234741 -0.37348465 -0.01566882]
 [ 0.32653273 -0.01834402 -0.021174   -0.51817316 -0.62468671]
 [-0.80588671 -0.36789508  0.04643859  0.61943541 -0.53806488]
 [ 2.93593126 -0.69004494 -0.47137816  0.12053677 -0.48416037]
 [-0.4905008  -0.57219604 -0.04207681  0.01958389 -0.42908113]
 [ 3.78864503 -0.53323093  1.1915708  -0.49068375 -0.52926639]
 [-0.60980371 -0.75308563 -0.05213356  0.0428904  -0.14504722]
 [-0.16953561 -0.39234702  0.27657455 -0.11061581 -0.56444561]
 [-0.8463893  -0.63011113  3.63679342 -0.62080285 -0.42169165]
 [-0.47303817 -0.49323546 -0.09088068  0.25152904  0.08020889]
 [ 0.80701979 -0.42390798 -0.36120765 -0.61084388  0.05903316]
 [ 0.01613834  1.68208216 -0.40629726  0.01387101 -0.21465832]
 [-0.70047927  0.3192561  -0.40649293 -0.278265    1.59803494]
 [-0.58579759  0.63974188 -0.30632817 -0.73458764  0.36907974]
 [-0.33645727 -0.6682984  -0.47668146 -0.58047973 -0.06466875]
 [ 0.04794688 -0.19653032 -0.42367974 -0.62279416 -0.52176166]
 [-0.46762718 -0.14658714 -0.06626418 -0.62638074 -0.59101045]
 [-0.26717878 -0.74766193 -0.43836784 -0.13471631 -0.00827002]
 [-0.73436477 -0.71797891  0.14531984 -0.55235355  0.08664755]
 [-0.71590253  1.67991241 -0.51226844  5.72808812  0.72270818]
 [ 3.71207711 -0.28492749 -0.2278289  -0.18357289  0.34113404]
 [-0.67664006 -0.20963084 -0.50484665 -0.31069039 -0.46700461]
 [-0.53898529  0.05232294 -0.41035319 -0.16600171 -0.04830827]
 [-0.2683504   0.27412267 -0.35905323  0.01430565 -0.27247715]
 [ 0.53742837 -0.27035874 -0.47947923  0.87941214 -0.26043486]
 [-0.60540113 -0.44479725 -0.29183271 -0.40951203 -0.35081291]
 [ 0.64844448 -0.63216681 -0.33063757  0.16436697 -0.05063884]
 [ 0.33706031 -0.16738372 -0.4475467   0.75645182 -0.33417581]
 [ 0.45178017 -0.56903064 -0.50104307  0.22533571 -0.01918241]
 [-0.42683168 -0.26789496 -0.39234708  0.7011493  -0.60389686]
 [-0.08325328 -0.72752582 -0.2182956  -0.01999243  5.9720231 ]
 [ 0.04691299 -0.31064941 -0.24557816 -0.28328581 -0.30854683]
 [-0.53836747  1.04847275  0.55880294 -0.16126379 -0.56614615]
 [ 0.50269988  0.76553835 -0.42063521 -0.40417603 -0.15618   ]
 [-0.59404474  5.58902219 -0.33012942 -0.34423542 -0.29032148]
 [-0.55955177  0.33418566 -0.45930391 -0.64770575  2.00482571]
 [-0.51800154 -0.38990055  0.11130411 -0.58372676 -0.44886474]]

Mean of each column: [4.41868764e-16 8.99280650e-17 2.19269047e-16 1.04360964e-16
 5.77315973e-17]

Standard deviation of each column:  [1. 1. 1. 1. 1.]
```

Question 3

In [17]:

```
# part a

# creating a Vandermonde matrix for N=3

def vandermonde (N):
    vec = np.arange (N) +1
    vec1 = np.arange(N)
    vec = vec.reshape(N,1)
    Matrix = (pow(vec,vec1))
    print(Matrix)
    return Matrix

n = 12
vander = n     #storing N=12 in a variable named 'vander'.
print('Vandermonde matrix is:')
```

```
Matrix_1 = vandermonde(vander)
```

```
Vandermonde matrix is:
[[          1           1           1           1           1           1
            1           1           1           1           1           1]
 [          1           2           4           8          16          32
           64         128         256         512        1024        2048]
 [          1           3           9          27          81         243
          729        2187        6561       19683       59049      177147]
 [          1           4          16          64         256        1024
         4096       16384       65536      262144     1048576     4194304]
 [          1           5          25         125         625        3125
        15625       78125      390625     1953125     9765625    48828125]
 [          1           6          36         216        1296        7776
        46656      279936     1679616    10077696    60466176   362797056]
 [          1           7          49         343        2401       16807
       117649      823543     5764801    40353607   282475249  1977326743]
 [          1           8          64         512        4096       32768
       262144     2097152    16777216   134217728  1073741824           0]
 [          1           9          81         729        6561       59049
       531441     4782969    43046721   387420489  -808182895  1316288537]
 [          1          10         100        1000       10000      100000
      1000000    10000000   100000000  1000000000  1410065408  1215752192]
 [          1          11         121        1331       14641      161051
      1771561    19487171   214358881 -1937019605   167620825  1843829075]
 [          1          12         144        1728       20736      248832
      2985984    35831808   429981696   864813056  1787822080   -20971520]]
```

In [18]:
```
# part b

x = np.ones(12)
b = np.dot(Matrix_1,x)
print(b)
```

```
[1.20000000e+01 4.09500000e+03 2.65720000e+05 5.59240500e+06
 6.10351560e+07 4.35356467e+08 2.30688120e+09 1.22713351e+09
 9.43953692e+08 3.73692871e+09 3.10225064e+08 3.10073456e+09]
```

In [19]:
```
# part c

# Solving the linear system the naïve way

Matrix_1_inv = np.linalg.inv(Matrix_1)
Sol = np.dot(Matrix_1_inv,b)
print(Sol)

# The result is almost ones(12) with slight variation maybe due to floating points instability while inversing
```

```
[0.99620819 1.00462723 0.99909973 1.00006104 0.99999666 1.00000048
 0.99999995 1.         1.         1.         1.         1.        ]
```

In [20]:
```
# d solve using numpy

Sol_inbuilt = np.linalg.solve(Matrix_1,b)

print(Sol_inbuilt)
```

```
[0.99998827 1.00002951 0.99997139 1.00001427 0.99999595 1.00000068
 0.99999993 1.         1.         1.         1.         1.        ]
```

In [21]:
```
# The solution using .solve() seems more accurate but let's verify that using some statistics

Diff_solve = x - Sol
Diff_solve_inbuilt = x - Sol_inbuilt
print(np.std(Diff_solve) , np.std(Diff_solve_inbuilt) )
# clearly the inbuilt function method's solution is more closer to ones(12)
```

```
0.0017465029209462444 1.3062999784531137e-05
```

In [22]:
```
# https://github.com/vinayak-sindagi/DSTT/tree/main/Vinayak%20Sindagi_24-27-30
```