Assn2_Parth_Dethaliya_24-27-29

October 8, 2024

[1]:
```
# Name: Parth Jilubhai Dethaliya
# Reg No: 24-27-29
# Programme: M.Tech. Data Science
# Assignment Number: 2
```

[2]:
```
import numpy as np
def print_var_info(var):
    print(f"values    : \n{var}")
    print(f"Shape     : {var.shape}")
    print(f"Dimension : {var.ndim}")
```

[3]:
```
## Q1
# a
var1 = np.arange(31) # including 0 and 30 as well, total 31 values

print_var_info(var1)
```

[4]:
```
# b
# Since 31 values are present in var1 (5,6) or (6,5) shapes may not work so,
var2 = var1.reshape(31,1)

print_var_info(var2)
```

[5]:
```
# c
# Similarly 3d shape
var3 = var2.reshape(1,31,1) # or var1.reshape() both would result in same

print_var_info(var3)
```

[6]:
```
# d

var2[1,0] = -1

print(var1)
print(var3)
```

The values in var1 as well as var3 changes because reshapes doesn't returns copy rather it returns view (in most of the cases).

[7]:
```python
# e i) Sum var3 over its second dimension
print(f"Shape     : {var3.shape}")

sum_var3_ax1 = np.sum(var3, axis=1)
print("Sum over second dimension:", sum_var3_ax1) # Second dimension contains
    ↪all the 31 values so it sums over it

# e ii) Sum var3 over its 3rd dimension

sum_var3_ax2 = np.sum(var3, axis=2)
print("Sum over second dimension:", sum_var3_ax2) # it will sum between columns
    ↪but since we only have 31 rows and
# single column so it will be same thing but with dimension 2 (as it has
    ↪reduced that 2nd axis by summin over it)

# e iii) Sum var3 over its 1st & 3rd dimension


sum_var3_ax1_3 = np.sum(var3, axis=(0,2))
print("Sum over second dimension:", sum_var3_ax1_3) # it willfirst sum over
    ↪axis 0 which is between the depth, but since
# there exists only one depth so the output information may not change, only
    ↪axis 0 will be reduces
# then it sums over axis 2, again same thing will happen as mention in "e (ii)"
    ↪and the dimension will be 1 now
```

Conclusion regarding output shape: if an array is of shape (a,b,c), any_operation(axis=0) returns (b,c) (eliminates the given axis) example: i) any_operation(axis=1) returns (a,c) ii) any_operation(axis=(0,1) returns (b) likewise...

[8]:
```python
# f i)
print("Second row")
print(var2[1,:])

# f ii)
print("\nLast column")
print(var2[:,-1]) # Only 1 column exists so the output will with full data but
    ↪1d

# f iii) Top right 2x2
# Since it's not possible with shape (31,1) so creating new variable with (5,6)
    ↪shape to do this
var4 = np.arange(30).reshape(5,6)
print("\nvar4: ")
print_var_info(var4)

print("\n")
```

```python
print("Top right 2x2: \n")

print(var4[:2,-2:])
```

[9]:
```python
# Q2
# a
arr = np.arange(10)
print("original array  : ", arr)
arr2 = arr + 1
print("Broadcasted + 1 : ",arr2)
```

[10]:
```python
# b 10x10 matrix
column_vector = arr2.reshape(10,1)
arr3 = column_vector + arr2
```

[11]:
```python
arr3
```

[12]:
```python
# c

import numpy.random as npr
data = np.exp(npr.randn ( 50 , 5 ) )
print(data)
```

[13]:
```python
std = np.std(data,axis=0)# for each column that means we have to compute␣
↪between rows
mean = np.mean(data,axis=0)
print(std)
print(mean)
```

[14]:
```python
normalized = (data-mean)/std
print(normalized)
```

[15]:
```python
std = np.std(normalized,axis=0)
mean = np.mean(normalized,axis=0)
print("Standard Deviation : ",std)
print("Mean               : ",mean)
# Mean 0, Std = 1
```

[16]:
```python
# 3 Vandermonde matrix

N = 12
def vandermonde(N):
    vec = np.arange (N) +1
    vector = np.arange(N) + 1
    v2 = np.arange(N)  # Power Vector
    output = vector.reshape(N,1)**v2
    return output
```

```
def printMat(Mat):
    for row in Mat:
        print(" ".join(f"{elem}" for elem in row))

Vector = vandermonde(N)
printMat(Vector)
```

[17]:
```
# b

x = np.ones(12)

b = np.dot(Vector,x)
print(b)
```

[18]:
```
# c naive solution

Vector_inv = np.linalg.inv(Vector)
Sol = np.dot(Vector_inv,b)
print(Sol)
# The result is almost ones(12) with slight variation maybe due to floating␣
↪points instability while inversing Vector
```

[19]:
```
# d solve using numpy

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

print(Sol_inbuilt)
```

[20]:
```
# 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)
```

[21]:
```
#https://github.com/PARTH1D/Parth_24-27-29/blob/main/
↪Assn2_Parth_Dethaliya_24-27-29.ipynb
```

[ ]:
```

```