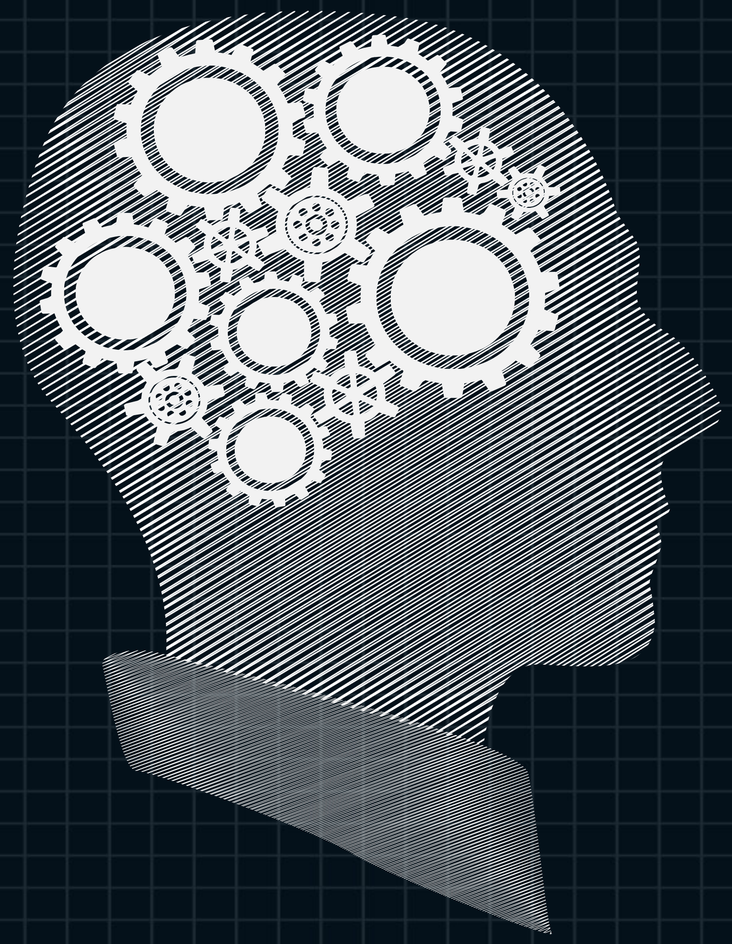


NumPy Master Class

Lecture.9
Sum, Prod, Diff and Statistics

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Summation

numpy.cumsum(*a*, *axis=None*, *dtype=None*, *out=None*)
ndarray.cumsum(*axis=None*, *dtype=None*, *out=None*)

```
import numpy as np

a = np.arange(5)
print("ndarray: ", a)

cumsum = np.cumsum(a)
print("cumsum: ", cumsum)
```

```
ndarray:  [0 1 2 3 4]

cumsum:  [ 0  1  3  6 10]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Summation

```
import numpy as np

a = np.arange(3*3).reshape((3, 3))
print("ndarray: {}\n{}".format(a.shape, a))
```

```
ndarray: (3, 3)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
cumsum = np.cumsum(a)
print("cumsum: {}\n{}".format(cumsum.shape, cumsum))
```

```
cumsum: (9,)
[ 0  1  3  6 10 15 21 28 36]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Summation

```
import numpy as np

a = np.arange(2*3*4).reshape((2, 3, 4))
print("ndarray: {}\n{}".format(a.shape, a))
```

```
ndarray: (2, 3, 4)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

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

```
cumsum = np.cumsum(a)
print("cumsum: {}\n{}".format(cumsum.shape, cumsum))
```

```
cumsum: (24,)
[  0   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120 136 153
 171 190 210 231 253 276]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Summation

```
import numpy as np

a = np.arange(3*4).reshape((3, 4))
print("ndarray: {}\n{}".format(a.shape, a))
```

```
ndarray: (3, 4)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
cumsum = np.cumsum(a, axis=0)
print("cumsum: {}\n{}"\
      .format(cumsum.shape,
              cumsum))
```

```
cumsum: (3, 4)
[[ 0  1  2  3]
 [ 4  6  8 10]
 [12 15 18 21]]
```

```
cumsum = np.cumsum(a, axis=1)
print("cumsum: {}\n{}"\
      .format(cumsum.shape,
              cumsum))
```

```
cumsum: (3, 4)
[[ 0  1  3  6]
 [ 4  9 15 22]
 [ 8 17 27 38]]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Product

numpy.prod(*a*, *axis=None*, *dtype=None*, *out=None*, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)
numpy.cumprod(*a*, *axis=None*, *dtype=None*, *out=None*)

```
import numpy as np

a = np.arange(1, 5)
print("ndarray: {}\n{}\n".format(a.shape, a))


prod = np.prod(a)
cumprod = np.cumprod(a)

print("prod: {}\n{}".format(prod.shape, prod))
print("cumprod: {}\n{}".format(cumprod.shape, cumprod))
```

```
ndarray: (4,)
[1 2 3 4]

prod: ()
24
cumprod: (4,)
[ 1  2  6 24]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Product

```
import numpy as np

a = np.arange(1, 1+12).reshape((3, 4))
print("ndarray: {}\n{}\n".format(a.shape, a))


prod = np.prod(a, axis=0)
cumprod = np.cumprod(a, axis=0)

print("prod(axis=0): {}\n{}".format(prod.shape, prod))
print("cumprod(axis=0): {}\n{}".format(cumprod.shape, cumprod))
```

```
ndarray: (3, 4)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]


prod(axis=0): (4,)
[ 45 120 231 384]
cumprod(axis=0): (3, 4)
[[  1   2   3   4]
 [  5  12  21  32]
 [ 45 120 231 384]]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Difference

numpy.diff(*a, n=1, axis=-1, prepend=<no value>, append=<no value>*)

```
import numpy as np

a = np.random.randint(0, 10, (5, ))
print("ndarray: {}\n{}\n".format(a.shape, a))


diff = np.diff(a)

print("diff: {}\n{}".format(diff.shape, diff))
```

```
ndarray: (5,)
[2 5 4 8 1]


diff: (4,)
[ 3 -1  4 -7]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Differentiation of Discrete-time Signal

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f'[n] = \frac{f[n+1] - f[n]}{1} = f[n+1] - f[n]$$

$$g[n] = f[n+1] - f[n]$$

Lecture.9 Sum, Prod, Diff and Statistics

- Sum, Prod, and Diff

Differentiation of Discrete-time Signal

```
import numpy as np
np.random.seed(0)

a = np.random.randint(0, 10, (4, 4))
print("ndarray: {}\n{}\n".format(a.shape, a))
```

```
ndarray: (4, 4)
[[5 0 3 3]
 [7 9 3 5]
 [2 4 7 6]
 [8 8 1 6]]
```

```
diff = np.diff(a, axis=0)
print("diff: {}\n{}"\
      .format(diff.shape,
              diff))
```

```
diff: (3, 4)
[[ 2  9  0  2]
 [-5 -5  4  1]
 [ 6  4 -6  0]]
```

```
diff = np.diff(a, axis=1)
print("diff: {}\n{}"\
      .format(diff.shape,
              diff))
```

```
diff: (4, 3)
[[-5  3  0]
 [ 2 -6  2]
 [ 2  3 -1]
 [ 0 -7  5]]
```

Lecture.9 Sum, Prod, Diff and Statistics

- Statistics

Mean and Median

numpy.mean(*a, axis=None, dtype=None, out=None, keepdims=<no value>, *, where=<no value>*)
ndarray.mean(*axis=None, dtype=None, out=None, keepdims=False, *, where=True*)

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\bar{x} = \sum_{i=1}^{n} w_i x_i, \quad where \quad \sum_{i=1}^{n} w_i = 1$$

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Mean and Median

```
import numpy as np
np.random.seed(0)

x = np.random.randint(1, 10, (5,))
w = np.array([1, 2, 3, 4, 5])

print(np.average(x, weights=w))    5.066666666666666

print(np.sum(w*x)/np.sum(w))       5.066666666666666
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Mean and Median

numpy.median(*a*, *axis=None*, *out=None*, *overwrite_input=False*, *keepdims=False*)

```
import numpy as np

x = np.arange(9)
median = np.median(x)

print(x)
print(median)
```

```
[0 1 2 3 4 5 6 7 8]
4.0
```

```
import numpy as np

x = np.arange(10)
median = np.median(x)

print(x)
print(median)
```

```
[0 1 2 3 4 5 6 7 8 9]
4.5
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Mean and Median

```
import numpy as np

x = np.random.randint(1, 10, (100,))

mean = np.mean(x)
median = np.median(x)

print("mean/median: {} / {}".format(mean, median))
```

```
mean/median: 4.66 / 4.0
```

```
x = np.append(x, 1000)

mean = np.mean(x)
median = np.median(x)

print("mean/median: {} / {}".format(mean, median))
```

```
mean/median: 14.514851485148515 / 4.0
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Variance and Standard Deviation

numpy.var(*a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>, *, where=<no value>*)
ndarray.var(*axis=None, dtype=None, out=None, ddof=0, keepdims=False, *, where=True*)

numpy.std(*a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>, *, where=<no value>*)
ndarray.std(*axis=None, dtype=None, out=None, ddof=0, keepdims=False, *, where=True*)

```
import numpy as np

scores = np.random.normal(loc=10,
                          scale=5,
                          size=(100, ))


var = scores.var()
std = scores.std()

print("variance: ", var)
print("standard deviation: ", std, '\n')
```

```
  variance:  24.165158618655738
  standard deviation:  4.915807016010264
```

```
print("square of std: ", std**2)
print("square root of var: ", var**0.5)
```

```
  square of std:  24.165158618655738
  square root of var:  4.915807016010264
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Standardization

```
import numpy as np

means = [50, 60, 70]
stds = [3, 5, 10]
n_student, n_class = 100, 3

scores = np.random.normal(loc=means,
                          scale=stds,
                          size=(n_student, n_class))
scores = scores.astype(np.float32)

print("shape of scores: ", scores.shape)
print("dtype of scores: ", scores.dtype)
```

```
 shape of scores:  (100, 3)
 dtype of scores:  float32
```

```
means = scores.mean(axis=0)
stds = scores.std(axis=0)

print("means before stdz: \n", means)
print("stds before stdz: \n", stds, '\n')
```

```
  means before stdz:
  [49.799522 60.17915  69.64157 ]
  stds before stdz:
  [ 3.255716  5.217268 10.325207]
```

```
score_stdz = (scores - means)/stds # standardization

means_stdz = score_stdz.mean(axis=0)
stds_stdz = score_stdz.std(axis=0)

print("means after stdz: \n", means_stdz)
print("stds after stdz: \n", stds_stdz)
```

```
  means after stdz:
  [ 4.6849252e-07 -4.0593745e-06  1.4561415e-06]
  stds after stdz:
  [1.         1.0000002 1.0000001]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Max Values and Indices

numpy.amax(*a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>*)
ndarray.max(*axis=None, out=None, keepdims=False, initial=<no value>, where=True*)

numpy.argmax(*a, axis=None, out=None*)
ndarray.argmax(*axis=None, out=None*)

```
import numpy as np

a = np.random.randint(0, 100, (10, ))

M = np.max(a)
M_idx = np.argmax(a)

print(f"ndarray: \n{a}")
print(f"M/M_idx: {M}/{M_idx}")
```

```
ndarray:
[88 85 28 58 85 78 65 76 11 25]
M/M_idx: 88/0
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Max Values and Indices

```
import numpy as np

means = [50, 60, 70]
stds = [3, 5, 10]
n_student, n_class = 100, 3

scores = np.random.normal(loc=means,
                          scale=stds,
                          size=(n_student, n_class))
scores = scores.astype(np.float32)


scores_max = np.max(scores, axis=0)
scores_max_idx = np.argmax(scores, axis=0)

print("Max scores: ", scores_max)
print("Max indices: ", scores_max_idx)

scores_max_idx = np.argmax(scores, axis=1)
print("Max subjects: ", scores_max_idx)
```

```
Max scores:  [57.4916  70.66934 87.54986]
Max indices:  [60 96 62]

Max subjects:  [2 1 2 2 2 1 2 2 2 2 2 2 2 2 1 1 1
 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 1 1 1 2 2 1 1 1 2
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 2
 2 2 2 1 2 2 2 2]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Min Values and Indices

```
numpy.amin(a, axis=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)
ndarray.min(axis=None, out=None, keepdims=False, initial=<no value>, where=True)
```

```
numpy.argmin(a, axis=None, out=None)
ndarray.argmin(axis=None, out=None)
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Min-max Normalization

```
import numpy as np

means = [50, 60, 70]
stds = [3, 5, 10]
n_student, n_class = 100, 3

scores = np.random.normal(loc=means,
                          scale=stds,
                          size=(n_student, n_class))
scores = scores.astype(np.float32)

scores_max = np.amax(scores, axis=0)
scores_min = np.amin(scores, axis=0)

scores_mM_norm = (scores - scores_min)/(scores_max - scores_min)

scores_max = np.amax(scores_mM_norm, axis=0)
scores_min = np.amin(scores_mM_norm, axis=0)

print(f"Max scores: {scores_max}")    Max scores: [1. 1. 1.]
print(f"min scores: {scores_min}")    min scores: [0. 0. 0.]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```python
import numpy as np

u = np.random.randint(0, 10, (10, ))
v = np.random.randint(0, 10, (10, ))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")

maximum = np.maximum(u, v)
minimum = np.minimum(u, v)

print(f"maximum: {maximum.shape}\n{maximum}")
print(f"minimum: {minimum.shape}\n{minimum}")
```

```
u: (10,)
[3 2 6 9 3 8 5 4 5 0]
v: (10,)
[4 6 4 4 8 6 5 1 8 5]


maximum: (10,)
[4 6 6 9 8 8 5 4 8 5]
minimum: (10,)
[3 2 4 4 3 6 5 1 5 0]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```python
import numpy as np

u = np.random.randint(0, 10, (3, 4))
v = np.random.randint(0, 10, (3, 4))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")
```

```
u: (3, 4)
[[1 2 5 4]
 [9 5 9 6]
 [3 1 5 3]]
v: (3, 4)
[[3 0 1 8]
 [0 9 1 9]
 [3 3 1 0]]
```

```python
maximum = np.maximum(u, v)
minimum = np.minimum(u, v)

print(f"maximum: {maximum.shape}\n{maximum}")
print(f"minimum: {minimum.shape}\n{minimum}")
```

```
maximum: (3, 4)
[[3 2 5 8]
 [9 9 9 9]
 [3 3 5 3]]
minimum: (3, 4)
[[1 0 1 4]
 [0 5 1 6]
 [3 1 1 0]]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```
import numpy as np

u = np.random.randint(0, 10, (10, ))
v = np.random.randint(0, 10, (10, ))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")
```

```
u: (10,)
[5 2 9 5 9 2 4 1 3 7]
v: (10,)
[2 1 6 2 4 9 5 4 7 0]
```

```
maximum = np.zeros_like(u)
maximum[u >= v] = u[u >= v]
maximum[u < v] = v[u < v]

print(f"np.maximum: \n{np.maximum(u, v)}")
print(f"maximum: \n{maximum}")
```

```
np.maximum:
[5 2 9 5 9 9 5 4 7 7]
maximum:
[5 2 9 5 9 9 5 4 7 7]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```
import numpy as np

u = np.random.randint(0, 10, (10, ))
v = np.random.randint(0, 10, (10, ))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")

up_vals = np.full_like(u, fill_value=100)
down_vals = np.full_like(u, fill_value=-100)

print(np.where(u > v, up_vals, down_vals))
```

```
u: (10,)
[5 2 0 3 1 1 8 5 8 5]
v: (10,)
[5 3 8 4 0 7 5 3 2 1]

[-100 -100 -100 -100  100 -100  100  100  100  100]
```

Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```python
import numpy as np

u = np.random.randint(0, 10, (10, ))
v = np.random.randint(0, 10, (10, ))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")

maximum = np.maximum(u, v)
maximum_where = np.where(u > v, u, v)

print(f"maximum: {maximum.shape}\n{maximum}")
print(f"maximum(where): {maximum_where.shape}\n{maximum_where}")
```

```
u: (10,)
[4 9 9 5 9 1 4 6 9 8]
v: (10,)
[4 4 9 9 3 2 5 0 9 2]

maximum: (10,)
[4 9 9 9 9 2 5 6 9 8]
maximum(where): (10,)
[4 9 9 9 9 2 5 6 9 8]
```

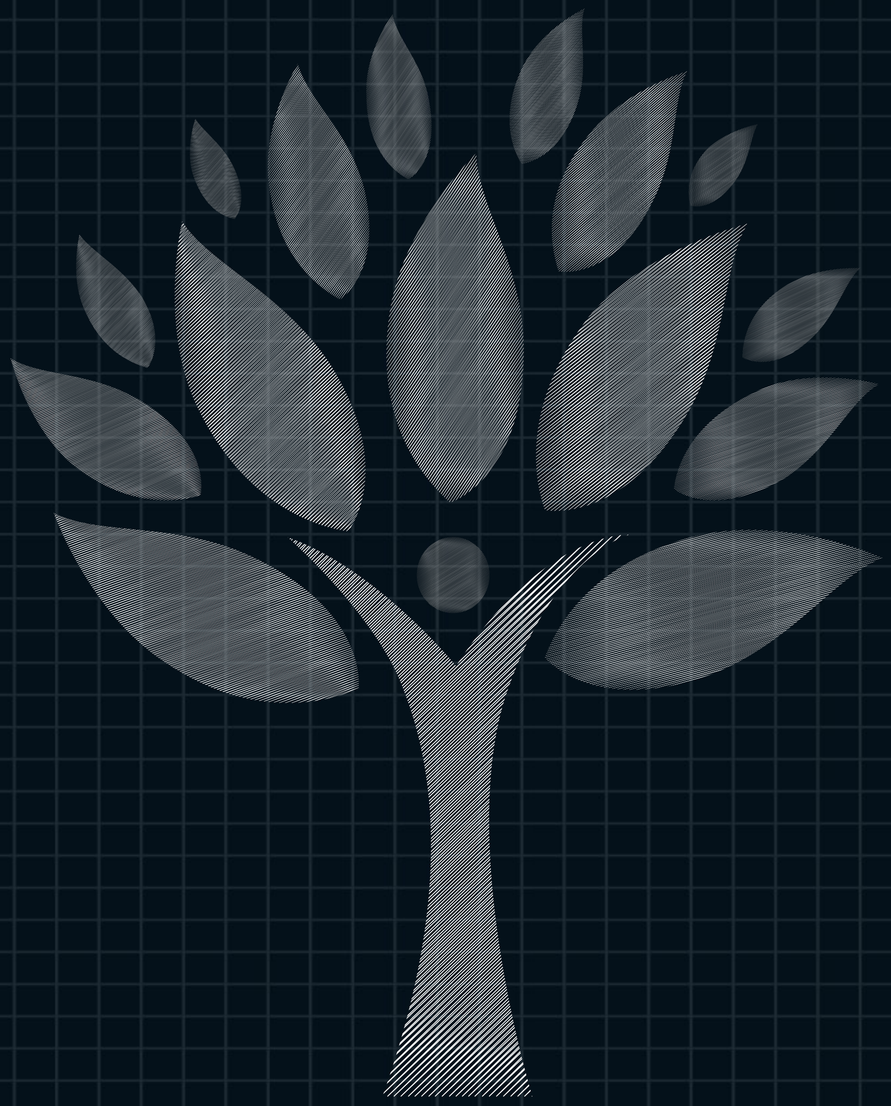
Lecture.9 Sum, Prod, Diff and Statistics - Statistics

Maximum and Minimum

```
import numpy as np

u = np.random.randint(0, 10, (10, ))
v = np.random.randint(0, 10, (10, ))
print(f"u: {u.shape}\n{u}")
print(f"v: {v.shape}\n{v}\n")

minimum = np.minimum(u, v)

minimum_where = np.where(u > v, v, u)
minimum_where2 = np.where(u < v, u, v)

print(f"minimum: {minimum.shape}\n{minimum}")
print(f"minimum(where): {minimum_where.shape}\n{minimum_where}")
print(f"minimum(where): {minimum_where2.shape}\n{minimum_where2}")
```

```
u: (10,)
[0 2 1 0 1 5 1 2 0 3]
v: (10,)
[4 4 5 9 4 5 0 7 8 2]
```

```
minimum: (10,)
[0 2 1 0 1 5 0 2 0 2]
minimum(where): (10,)
[0 2 1 0 1 5 0 2 0 2]
minimum(where): (10,)
[0 2 1 0 1 5 0 2 0 2]
```



NumPy Master Class

Lecture.9
Sum, Prod, Diff and Statistics