[11pt]article
[breakable]tcolorbox parskip
graphicx caption nocaption format=nocaption,aboveskip=0pt,belowskip=0pt
float figureH xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym
iftex fontspec unicode-math
fancyvrb grffile [Export]adjustbox max size=0.90.9
hyperref titling longtable booktabs array calc [inline]enumitem [normalem]ulem soul mathrsfs
urlcolorrgb0,.145,.698 linkcolorrgb.71,0.21,0.01 citecolorrgb.12,.54,.11
ansi-blackHTML3E424D ansi-black-intenseHTML282C36 ansi-redHTMLE75C58 ansi-red-intenseHTMLB22B31
ansi-greenHTML00A250 ansi-green-intenseHTML007427 ansi-yellowHTMLDDB62B ansi-yellow-intenseHTMLB27D12
ansi-blueHTML208FFB ansi-blue-intenseHTML0065CA ansi-magentaHTMLD160C4 ansi-magenta-intenseHTMLA03196
ansi-cyanHTML60C6C8 ansi-cyan-intenseHTML258F8F ansi-whiteHTMLC5C1B4 ansi-white-intenseHTMLA1A6B2
ansi-default-inverse-fgHTMLFFFFFF ansi-default-inverse-bgHTML000000
outerrorbackgroundHTMLFFDFDF
HighlightingVerbatimcommandchars=
{}
incolorHTML303F9F outcolorHTMLD84315 cellborderHTMLCFCFCF cellbackgroundHTMLF7F7F7
breaklinks=true, colorlinks=true, urlcolor=urlcolor, linkcolor=linkcolor, citecolor=citecolor,
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

StatisticsUsingNumpyAndScipy (1)

February 22, 2025

0.1 Basic Statistical Functions using Numpy and ScipyBasic Statistical Functions using Numpy and Scipy

Using NumPy Module

[commandchars=\\\{\},codes*=] [rgb]0.00,0.50,0.00**import** [rgb]0.00,0.00,1.00**numpy** [rgb]0.00,0.50,0.00**as** [rgb]0.00,0.00,1.00**np**

Statistical Operation

1. mean()- This function basically calculates the average from an array of data.

[commandchars=\\\{\},codes*=] x[rgb]0.40,0.40,0.40=np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.402,[rgb]0.40,0.40,0.403,[rgb]0.40,0.40,0.404,[rgb]0.40,0.40,0.405]) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Output of list items in x[rgb]0.67,0.36,0.12\n[rgb]0.73,0.13,0.13") [rgb]0.00,0.50,0.00print(x[rgb]0.40,0.40,0.40.mean())

[commandchars=\\\{\},codes*=] Output of list items in x
3.0

Here below, first we have created the 2D array named array1. We then calculated the mean using np.mean().

np.mean(array1) - calculates the mean over the entire array

np.mean(array1, axis=0) - calculates the mean along vertical axis

np.mean(array1, axis=1) calculates the mean along horizontal axis

[commandchars=\\\{\},codes*=] [rgb]0.24,0.48,0.48*# create a 2D array* array1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([[[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.403], [[rgb]0.40,0.40,0.405, [rgb]0.40,0.40,0.407]])

[rgb]0.24,0.48,0.48*# calculate the mean of the entire array* result1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.mean(array1) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Entire Array: [rgb]0.73,0.13,0.13",result1) [rgb]0.24,0.48,0.48*# 4.0*

[rgb]0.24,0.48,0.48*# calculate the mean along vertical axis (axis=0)* result2 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.mean(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.400) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Along Vertical Axis:[rgb]0.73,0.13,0.13",result2) [rgb]0.24,0.48,0.48*# [3. 5.]*

[rgb]0.24,0.48,0.48*# calculate the mean along (axis=1)* result3 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.mean(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.401) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.1 Horizontal Axis :[rgb]0.73,0.13,0.13",result3) [rgb]0.24,0.48,0.48*# [2. 6.]*

[commandchars=\\\{\},codes*=] Entire Array: 4.0 Along Vertical Axis: [3. 5.] Along Horizontal Axis : [2. 6.]

2. median()- This function calculates the statistical median of the elements of the array given.

[commandchars=\\\{\},codes*=] x[rgb]0.40,0.40,0.40=np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.402,[rgb]0.40,0.40,0.403,[rgb]0.40,0.40,0.404,[rgb]0.40,0.40,0.405]) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Output of odd no. of list items in x[rgb]0.67,0.36,0.12\n[rgb]0.73,0.13,0.13") [rgb]0.00,0.50,0.00print(np[rgb]0.40,0.40,0.40.median(x))

[commandchars=\\\{\},codes*=] Output of odd no. of list items in x
3.0

[commandchars=\\\{\},codes*=] x[rgb]0.40,0.40,0.40=np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.401,[rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.403,[rgb]0.40,0.40,0.404,[rgb]0.40,0.40,0.405,[rgb]0.40,0.40,0.406]) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Output of even no. of list items in x[rgb]0.67,0.36,0.12\n[rgb]0.73,0.13,0.13") [rgb]0.00,0.50,0.00print(np[rgb]0.40,0.40,0.40.median(x))

[commandchars=\\\{\},codes*=] Output of even no. of list items in x
3.5

Calculation of the median is not just limited to 1D array. We can also calculate the median of the 2D array.

In a 2D array, median can be calculated either along the horizontal or the vertical axis individually, or across the entire array.

When computing the median of a 2D array, we use the axis parameter inside np.median() to specify the axis along which to compute the median.

If we specify,

axis = 0, median is calculated along vertical axis axis = 1, median is calculated along horizontal axis If we don't use the axis parameter, the median is computed over the entire array.

[breakable,size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=incolor, commandchars=\\\{\},codes*=] [rgb]0.24,0.48,0.48*# create a 2D array* array1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redarray([[[rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.404, [rgb]0.40,0.40,0.406], [[rgb]0.40,0.40,0.408, [rgb]0.40,0.40,0.4010, [rgb]0.40,0.40,0.4012], [[rgb]0.40,0.40,0.4014, [rgb]0.40,0.40,0.4016, [rgb]0.40,0.40,0.4018]])

[rgb]0.24,0.48,0.48*# compute median along horizontal axis* result1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redmedian(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.401)

[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Median along horizontal axis : red[rgb]0.73,0.13,0.13", result1)

[rgb]0.24,0.48,0.48*# compute median along vertical axis* result2 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redmedian(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.400)

[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Median along vertical axis:[rgb]0.73,0.13,0.13", result2)

[rgb]0.24,0.48,0.48*# compute median of entire array* result3 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redmedian(array1)

[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Median of entire array:[rgb]0.73,0.13,0.13", result3)

[commandchars=\\\{\},codes*=] Median along horizontal axis : [4. 10. 16.] Median along vertical axis: [8. 10. 12.] Median of entire array: 10.0

3.std()- This function calculates the statistical standard deviation of the elements of the array given

The standard deviation is a measure of the spread of the data in the array. It gives us the degree to which the data points in an array deviate from the mean.

Smaller standard deviation indicates that the data points are closer to the mean

Larger standard deviation indicates that the data points are more spread out.

[breakable,size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=incolor, commandchars=\\\{\},codes*=] [rgb]0.24,0.48,0.48*# create a numpy array* marks [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redarray([[rgb]0.40,0.40,0.4076, [rgb]0.40,0.40,0.4078, [rgb]0.40,0.40,0.4081, [rgb]0.40,0.40,0.4066, [rgb]0.40,0.40,0.4085])

[rgb]0.24,0.48,0.48*# compute the standard deviation of marks* std_marks [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.std(marks) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Standard Deviation of the above marks list[rgb]0.73,0.13,0.13") [rgb]0.00,0.50,0.00print(std_marks)

[commandchars=\\\{\},codes*=] 6.368673331236263

In a 2D array, standard deviation can be calculated either along the horizontal or the vertical axis individually, or across the entire array.

Similar to mean and median, when computing the standard deviation of a 2D array, we use the axis parameter inside np.std() to specify the axis along which to compute the standard deviation.

[breakable,size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=incolor, commandchars=\\\{\},codes*=] [rgb]0.24,0.48,0.48*# create a 2D array* array1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.redarray([[[rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.405, [rgb]0.40,0.40,0.409], [[rgb]0.40,0.40,0.403, [rgb]0.40,0.40,0.408, [rgb]0.40,0.40,0.4011], [[rgb]0.40,0.40,0.404, [rgb]0.40,0.40,0.406, [rgb]0.40,0.40,0.407]])

[rgb]0.24,0.48,0.48*# compute standard deviation along horizontal axis* result1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.std(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.401) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Standard deviation along horizontal axis: red[rgb]0.73,0.13,0.13", result1)

[rgb]0.24,0.48,0.48*# compute standard deviation along vertical axis* result2 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.std(array1, axis[rgb]0.40,0.40,0.40=[rgb]0.40,0.40,0.400) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Standard deviation along vertical axis: red[rgb]0.73,0.13,0.13", result2)

[rgb]0.24,0.48,0.48*# compute standard deviation of entire array* result3 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.std(array1) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Standard deviation of entire array:[rgb]0.73,0.13,0.13", result3)

[commandchars=\\\{\},codes*=] Standard deviation along horizontal axis: [2.86744176 3.29983165 1.24721913] Standard deviation along vertical axis: [0.81649658 1.24721913 1.63299316] Standard deviation of entire array: 2.7666443551086073

4. percentile()- In NumPy, we use the percentile() function to compute the nth percentile of a given array.

[commandchars=\{\},codes*=] [rgb]0.24,0.48,0.48# *create an array* array1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.403, [rgb]0.40,0.40,0.405, [rgb]0.40,0.40,0.407, [rgb]0.40,0.40,0.409, [rgb]0.40,0.40,0.4011, [rgb]0.40,0.40,0.4013, [rgb]0.40,0.40,0.4015, [rgb]0.40,0.40,0.4017, [rgb]0.40,0.40,0.4019])
[rgb]0.24,0.48,0.48# *compute the 25th percentile of the array* result1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.percentile(array1, [rgb]0.40,0.40,0.4025) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.1325th percentile:[rgb]0.73,0.13,0.13",result1)
[rgb]0.24,0.48,0.48# *compute the 75th percentile of the array* result2 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.percentile(array1, [rgb]0.40,0.40,0.4075) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.1375th percentile:[rgb]0.73,0.13,0.13",result2)

```
25th percentile: 5.5 75th percentile: 14.5
```

5. min()- Finds the minimum value of the array.
6. max()- Finds the maximum value of the array.

[commandchars=\{\},codes*=] [rgb]0.24,0.48,0.48# *create an array* array1 [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([[rgb]0.40,0.40,0.402,[rgb]0.40,0.40,0.406,[rgb]0.40,0.40,0.409,[rgb]0.40,0.40,0.4015, [rgb]0.40,0.40,0.4017,[rgb]0.40,0.40,0.4022,[rgb]0.40,0.40,0.4065,[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.4062])
[rgb]0.24,0.48,0.48# *find the minimum value of the array* min_val [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.min(array1)
[rgb]0.24,0.48,0.48# *find the maximum value of the array* max_val [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.max(array1)
[rgb]0.24,0.48,0.48# *print the results* [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Minimum value:[rgb]0.73,0.13,0.13", min_val) [rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Maximum value:[rgb]0.73,0.13,0.13", max_val)

```
Minimum value: 1 Maximum value: 65
```

7. cov()- This function calculates the statistical covarience between 2 matrices or arrays of same dimension

[commandchars=\{\},codes*=] x [rgb]0.40,0.40,0.40= np[rgb]0.40,0.40,0.40.array([[[rgb]0.40,0.40,0.400, [rgb]0.40,0.40,0.403, [rgb]0.40,0.40,0.404], [[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.404], [[rgb]0.40,0.40,0.403, [rgb]0.40,0.40,0.404, [rgb]0.40,0.40,0.405]])
[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Shape of array: [rgb]0.67,0.36,0.12\n[rgb]0.73,0.13,0.13", np[rgb]0.40,0.40,0.40.shape(x))
[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Covariance matrix of x: [rgb]0.67,0.36,0.12\n[rgb]0.73,0.13,0.13", np[rgb]0.40,0.40,0.40.cov(x))

```
Shape of array: (3, 3) Covariance matrix of x: [[4.33333333 2.83333333 2. ] [2.83333333 2.33333333
1.5 ] [2. 1.5 1. ]]
```

Using Scipy Module

[commandchars=\{\},codes*=] [rgb]0.00,0.50,0.00**from** [rgb]0.00,0.00,1.00**scipy** [rgb]0.00,0.50,0.00**import** stats

1. mode()- It takes an array as input and returns the mode along with their corresponding counts.

Mode is the data item in the dataset having largest occurance.:

[commandchars=\{\},codes*=] data [rgb]0.40,0.40,0.40= [[rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.402, [rgb]0.40,0.40,0.403, [rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.401, [rgb]0.40,0.40,0.405]
[rgb]0.00,0.50,0.00print([rgb]0.73,0.13,0.13"[rgb]0.73,0.13,0.13Mode:[rgb]0.73,0.13,0.13", stats[rgb]0.40,0.40,0.40.mode(data))

```
Mode: ModeResult(mode=1, count=3)
```