Helpful Functions and Procedures for the Exercises

by Prof. Dr. Peter Roßbach[1]

Functions

numpy.sort()

The sort() function returns a sorted version of the input array. It has the following parameters

```
numpy.sort(a, axis)
```

where

a : Array to be sorted

axis : The axis along which the array is to be sorted. By default, the last axis is used.

Examples:

```
>>> import numpy as np
>>> a = np.array([[3,7,5],[9,1,3],[4,2,1],[2,8,9]])
>>> print(a)
[[3 7 5]
 [9 1 3]
 [4 2 1]
 [2 8 9]]
>>> print(np.sort(a))   # uses axis=1 as last axis
[[3 5 7]
 [1 3 9]
 [1 2 4]
 [2 8 9]]
>>> print(np.sort(a, axis=0))
[[2 1 1]
 [3 2 3]
 [4 7 5]
 [9 8 9]]
>>> print(np.sort(a, axis=1))
[[3 5 7]
 [1 3 9]
 [1 2 4]
 [2 8 9]]
>>> print(np.sort(a[:,1]))   # sorts only the specified column
[1 2 7 8]
```

numpy.argsort()

The argsort() function works like sort, but it returns of indices instead of the values of an array. Applied to a vector, this can be used in conjunction with sliding to sort an array according a specific column.

Examples:

```
>>> print(np.argsort(a))   # uses axis=1 as last axis
[[0 2 1]
 [1 2 0]
 [2 1 0]
 [0 1 2]]
```

[1] This document is a reworked extract from the source: https://www.tutorialspoint.com/numpy/numpy_introduction.htm

```
>>> print(np.argsort(a, axis=0))
[[3 1 2]
 [0 2 1]
 [2 0 0]
 [1 3 3]]
>>> print(np.argsort(a[:,1]))   # sorts only the specified column
[1 2 0 3]
>>> print(a[np.argsort(a[:,1])])   # sorts the rows according to column 1
[[9 1 3]
 [4 2 1]
 [3 7 5]
 [2 8 9]]
```

numpy.amin() and numpy.amax()

These functions return the minimum and the maximum from the elements in the given array. If the axis parameter is not specified, the overall minimum/maximum is returned. Otherwise, the minima/maxima are calculated along the specified axis.

Examples:

```
>>> print(np.amin(a))
1
>>> print(np.amin(a, axis=0))
[2 1 1]
>>> print(np.amax(a))
9
>>> print(np.amax(a, axis=1))
 [7 9 4 9]
```

numpy.percentile()

Percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall. The function numpy.percentile() takes the following arguments.

```
numpy.percentile(a, q, axis)
```

where

a : Input array

q : The percentile to compute; must be between 0-100

axis : The axis along which the percentile is to be calculated

Examples:

```
>>> print(np.percentile(a, 50))
3.5
>>> print(np.percentile(a, 50, axis=0))
[3.5 4.5 4. ]
>>> print(np.percentile(a, 50, axis=1))
[5. 3. 2. 8.]
>>> print(np.percentile(a, 20))
2.0
>>> print(np.percentile(a, 20, axis=0))
[2.6 1.6 2.2]
>>> print(np.percentile(a, 20, axis=1))
[3.8 1.8 1.4 4.4]
```

numpy.median()

Median is defined as the value separating the higher half of a data sample from the lower half. The numpy.median() function returns the median of elements in the array. If the axis is mentioned, it is calculated along it.

Examples:

```
>>> print(np.median(a))
3.5
>>> print(np.median(a, axis=0))
[3.5 4.5 4. ]
>>> print(np.median(a, axis=1))
[5. 3. 2. 8.]
```

numpy.mean()

Arithmetic mean is the sum of elements along an axis divided by the number of elements. The numpy.mean() function returns the arithmetic mean of elements in the array. If the axis is mentioned, it is calculated along it.

Examples:

```
>>> print(np.mean(a))
4.5
>>> print(np.mean(a, axis=0))
[4.5 4.5 4.5]
>>> print(np.mean(a, axis=1))
[5.         4.33333333 2.33333333 6.33333333]
```

numpy.var()

Variance is the average of squared deviations, i.e., mean(abs(x - x.mean())**2). The numpy.var() function returns the variance of elements in the array. If the axis is mentioned, it is calculated along it.

An additional parameter $ddof$ (Delta Degrees of Freedom) represents the divisor used in the calculation is $(n - ddof)$, where n represents the number of elements. By default, $ddof$ is zero.

Examples:

```
>>> print(np.var(a))
8.416666666666666
>>> print(np.var(a, axis=0))
[7.25 9.25 8.75]
>>> print(np.var(a, axis=1))
[ 2.66666667 11.55555556  1.55555556  9.55555556]
>>> print(np.var(a, axis=0, ddof=1))
[ 9.66666667 12.33333333 11.66666667]
```

Note: For the calculation of the variance, in the lecture $ddof = 0$ is used.

numpy.std()

Standard deviation is the square root of the average of squared deviations from mean. The numpy.std() function returns the standard deviation of elements in the array. If the axis is mentioned, it is calculated along it.

Examples:

```
>>> print(np.std(a))
2.9011491975882016
>>> print(np.std(a, axis=0))
[2.6925824  3.04138127 2.95803989]
>>> print(np.std(a, axis=1))
[1.63299316 3.39934634 1.24721913 3.09120617]
>>> print(np.std(a, axis=0, ddof=1))
[3.10912635 3.51188458 3.41565026]
```

numpy.cov()

The covariance matrix is a square and symmetric matrix that describes the covariance between two or more variables. The numpy.cov() function returns the covariance matrix of two vectors.

Example:

```
>>> x = a[:,0]
>>> y = a[:,1]
>>> print(np.cov(x,y))
[[ 9.66666667 -9.        ]
 [-9.         12.33333333]]
```

The values on the main diagonal represent the variances of x and y, the others the covariance.

```
>>> print(np.cov(x,y)[0,1])   # extract the covariance
-9.0
>>> print(np.cov(x,y)[0,0])   # extract the variance of x
9.666666666666666
>>> print(np.cov(x,y)[1,1])   # extract the variance of y
12.333333333333332
```

Caution: In numpy, cov defaults to a "delta degree of freedom" of 1 while var defaults to a ddof of 0. This can be changed by the parameter ddof.

```
>>> print(np.cov(x,y, ddof=0))
 [[ 7.25 -6.75]
 [-6.75  9.25]]
```

numpy. corrcoef()

Analog to the covariance matrix, the correlation matrix is calculated by the function numpy. corrcoef().

Examples:

```
>>> print(np.corrcoef(x,y))
 [[ 1.         -0.82425939]
 [-0.82425939  1.        ]]
```

Plotting

Frequency Distribution, Histogram, and Cumulative Distribution Function of Data

Matplotlib can plot class frequency distributions as well as histograms and cumulative distribution functions of given data.

The hist() function takes a vector containing the data.

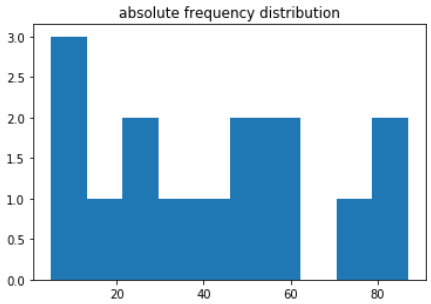
If the bins parameter is set, the number and boundaries of the intervals can be set. If bins is an integer, bins+1 bin edges are calculated. The default value for bins is 10. If bins is a list, then the edges and the sizes of the intervals can be individually determined.

By default the resulting plot shows the absolute frequency distribution of the data within the given bins/classes. If the density parameter is set to True, then a histogram is plotted (where the area under the plot sums up to 1).

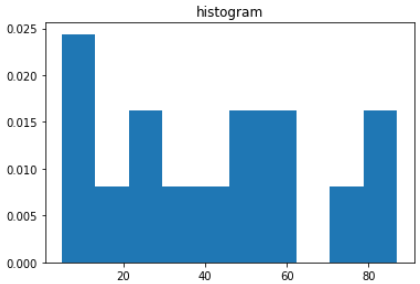
If the cumulative parameter is set to True, then a cumulative distribution function is computed where each bin gives the counts in that bin plus all bins for smaller values.
(Note: different to the lecture, here the steps are performed at the left hand side of the bin/class and not at the right hand side!)

Examples:

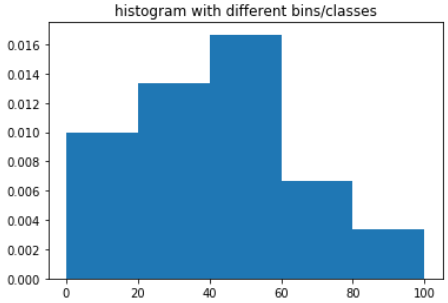
```
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
>>> plt.hist(a)
>>> plt.title("absolute frequency distribution")
>>> plt.show()
```



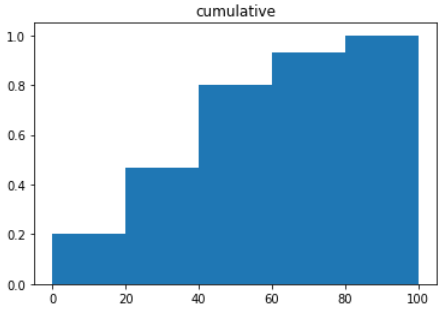

```
>>> plt.hist(a,density=True)
>>> plt.title("histogram")
>>> plt.show()
```




```
>>> plt.hist(a, bins = [0,20,40,60,80,100], density=True)
>>> plt.title("histogram with different bins/classes")
>>> plt.show()
```




```
>>> plt.hist(a, bins = [0,20,40,60,80,100], cumulative=True, density=True)
>>> plt.title("cumulative")
>>> plt.show()
```

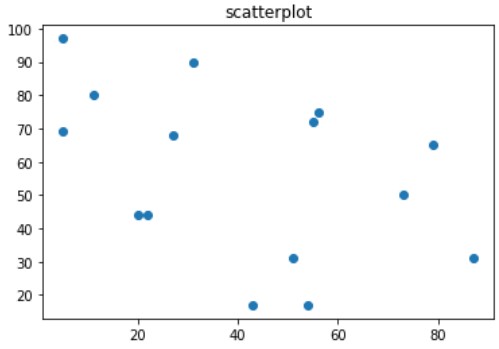


Scatter Plot

A commonly used plot type for paired date is the scatter plot. Here the data points are represented individually with a dot, circle, or other shape. Matplot has a built-in function to create scatterplots called scatter().

Example:

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
>>> b = np.array([44,31,69,17,75,50,72,17,80,44,31,97,65,90,68])
>>> plt.scatter(a,b)
>>> plt.title("scatterplot")
>>> plt.show()
```



Regression Analysis

Using Numpy, performing a regression analysis and plotting the results is very easy. The calculation of slope and intercept can be can be programmed in one line each and the predicted values in another line.

First, you have to load the data into two numpy vectors. Let's name them x for the independent variable and y for the dependent.

Second, you have to calculate the slope b by

$$b = \frac{Covariance(x, y)}{Variance(x)}$$

Third, you have to calculate the intercept *a* by

$$a = Mean(y) - b * Mean(x)$$

Finally, you can calculate the predictions via

$$\hat{y}_i = a + b * x_i$$

To plot the regression, you can create a scatterplot to plot x and y and you inject the regression line by plotting x and $\hat{y}$ via

```
plt.plot(x,y_hat, lw=4, c='orange')
```

The result using the data from the previous example looks like