Exercise Set 12

1 Skiing and Fondue [normal]

A large Swiss cheese company thinks about marketing its product on ski hills. The idea is that people who enjoy outdoor activities on the mountains in also enjoy cheese more, and thus one could sell Fondue to them better than to the general population. To test this, a marketing institute sampled 36 people (9 in each group) who ate/didn't eat fondue and went/didn't go skiing. Each person should rate how much they enjoyed their day on a scale between 1 and 10.

a) First, we want to investigate the effects of the individual factors. Create a table of the means for all four possible conditions, including marginal/partial means, and the total mean.

b) Create a two-factor ANOVA table, assuming the factors are independent. Can you say (using a level of significance of $\alpha = 0.05$), that each factor (skiing, and eating fondue) has a statistically significant effect? How different are the two effects?

Next week we will examine if there is an *interaction* between the two factors.

No skiing, no fondue	no skiing, fondue	skiing, no fondue	skiing, fondue
8	4	3	8
6	8	6	9
5	6	4	6
4	6	9	8
1	4	8	9
3	9	8	6
4	8	5	5
3	5	10	9
4	6	4	9

2 Exam-style Python questions [normal]

a) What output do you expect from the following code?

```
import numpy as np
myArray = np.array([3,2,7,5])
sortedArray = np.sort(myArray)
print(sortedArray[1])
```

b) In the next Jupyter cell, the code contines as follows. What output do you expect?

```
newArray = myArray+10
print(newArray)
```

c) In the next Jupyter cell, the code contines as follows. What output do you expect?

```
print(np.shape(myArray))
reshapedArray = np.reshape(myArray,(2,2))
print(np.shape(reshapedArray))
```

d) In the next Jupyter cell, the code contines as follows. What output do you expect and why?

```
print(myArray+np.array([1,2,3,4,5]))
```

e) You would like to calculate the unbiased estimator for the population standard deviation from the sample/data contained in `myArray` using the function `np.std`. The documentation (as found by running the code `np.std?` or similarly the numpy.org webpage) is given in the last page. How would you execute the function (i.e. with which arguments?)?

f) How would you find the minimum value of `myArray` (there are many possible solutions, any of them that works is fine)? (Write down the code but do not worry about precise syntax).

g) Name (at least) one difference between a standard Python list and a numpy array (for example how they respond differently to a certain operation).

3 Second exam question from 2024

On the following page you will find an exam question from 2024. The exam lasted three hours.

The questions on the next page accounted for 25 out of a total of 68 points - with proportional time allocation, this would correspond to about 66 minutes.

In the exam tables for the z-test, t-test, χ^2 test and F-test were provided (see the StatTables pdf on the moodle) .

Of course now that you do not have your condensed notes yet, you should use your course notes or favourite textbook, and can expect to take longer because of that.

5-sided die

You are studying a 5-sided die[1], with possible outcomes numbered 1,2,3,4,5. Initially, assume that all results have equal probability (i.e.[2] it is a fair die).

{1p} a) What is the probability of the outcomes "odd[3] number" and "number larger than 3"?

{2p} b) Are these two probabilities independent? Prove your answer any way you prefer.

{1p} c) What is the expectation value (for the mean) of this die?

{2p} d) In a game where playing once costs 10CHF, and the prize in one round corresponds to the thrown[4] number squared, calculate the expectation value for the overall gain or loss per round.

Now you try to build such a die, but you do not know if it is fair.

{2p} e) After throwing your die 5 times, you get the following results: [2,5,2,5,1]. Compute the mean, median and the unbiased estimator of the variance of this sample.

After throwing 200 times, the 5 possible results occur the following number of times:

Result	1	2	3	4	5
Number of occurences	48	35	38	33	46

{1p} f) Compute the difference between this outcome and the expectation values for each result for a fair die.

{1p} g) To test the hypothesis "this die is fair, each result has the same probability", which statistical test is suitable?

{4p} h) Perform this test, using a confidence level of 95% (significance level of 5%).

{1p} i) Was throwing 200 times sufficient to perform this test? Justify your statement.

A junior colleague of yours has written the following Python script to find the median of the data given in (e):

```
import numpy as np

def findMedian(inputArray):
    sortedArray = np.sort(inputArray)
    print("Sorted Array:")
    print(sortedArray)
    arrayLength = len(sortedArray)
    print("Length of the array:")
    print(arrayLength)
    medianPositionFloat = arrayLength/2+0.5
    medianPosition = int(medianPositionFloat)#transform to integer
    print("Position of the median:")
    print(medianPosition)
    median = sortedArray[medianPosition]
    return median
```

[1]dé [2]c'est-à-dire [3]impaire [4]lancé

```
myArray = np.array([2,5,2,5,1])
myMedian = findMedian(myArray)
print("The median:")
print(myMedian)
```

{3p} j) Which outputs do you expect from this code? *Note that* `"print"` *calls inside a function will also produce an output.*

{2p} k) Why does the script give the wrong median? How can you correct it by making only a small change to the code above?

{3p} l) Furthermore, this code (also with the small correction) would not work for input arrays with an even[5] number of elements. Describe in words or with code (precise syntax is not important) how you would write a script that can also process inputs with an even number of elements. *There are several possible solutions, some longer, some shorter, every solution that works correctly for both even and odd numbers of elements is acceptable.*

{2p} m) In general, if the mean and the median of a set of data *are not* the same, this implies that the data *are not* symmetric around the mean. If the mean and the median of a set of data *are* the same, does that *always* imply that the data *is* symmetric around the mean? Prove your answer any way you like.

[5] pair

Documentation of the np.std function

As printed when running `np.std?` after the code above.

```
Signature:
np.std(
    a,
    axis=None,
    dtype=None,
    out=None,
    ddof=0,
    keepdims=<no value>,
    *,
    where=<no value>,
)
Call signature:  np.std(*args, **kwargs)
Type:            _ArrayFunctionDispatcher
String form:     <function std at 0x734d0c46b6a0>
File:            /opt/jlab-env/lib/python3.12/site-packages/numpy/core/fromnumeric.py
Docstring:
Compute the standard deviation along the specified axis.

Returns the standard deviation, a measure of the spread of a distribution,
of the array elements. The standard deviation is computed for the
flattened array by default, otherwise over the specified axis.

Parameters
----------
a : array_like
    Calculate the standard deviation of these values.
axis : None or int or tuple of ints, optional
    Axis or axes along which the standard deviation is computed. The
    default is to compute the standard deviation of the flattened array.

    .. versionadded:: 1.7.0

    If this is a tuple of ints, a standard deviation is performed over
    multiple axes, instead of a single axis or all the axes as before.
dtype : dtype, optional
    Type to use in computing the standard deviation. For arrays of
    integer type the default is float64, for arrays of float types it is
    the same as the array type.
out : ndarray, optional
    Alternative output array in which to place the result. It must have
    the same shape as the expected output but the type (of the calculated
    values) will be cast if necessary.
ddof : int, optional
    Means Delta Degrees of Freedom.  The divisor used in calculations
    is ``N - ddof``, where ``N`` represents the number of elements.
    By default `ddof` is zero.
keepdims : bool, optional
    If this is set to True, the axes which are reduced are left
    in the result as dimensions with size one. With this option,
    the result will broadcast correctly against the input array.
```

```
    If the default value is passed, then ‘keepdims‘ will not be
    passed through to the ‘std‘ method of sub-classes of
    ‘ndarray‘, however any non-default value will be.  If the
    sub-class’ method does not implement ‘keepdims‘ any
    exceptions will be raised.

where : array_like of bool, optional
    Elements to include in the standard deviation.
    See ‘˜numpy.ufunc.reduce‘ for details.

    .. versionadded:: 1.20.0

Returns
-------
standard_deviation : ndarray, see dtype parameter above.
    If ‘out‘ is None, return a new array containing the standard deviation,
    otherwise return a reference to the output array.

See Also
--------
var, mean, nanmean, nanstd, nanvar
:ref:‘ufuncs-output-type‘

Notes
-----
The standard deviation is the square root of the average of the squared
deviations from the mean, i.e., ‘‘std = sqrt(mean(x))‘‘, where
‘‘x = abs(a - a.mean())**2‘‘.

The average squared deviation is typically calculated as ‘‘x.sum() / N‘‘,
where ‘‘N = len(x)‘‘. If, however, ‘ddof‘ is specified, the divisor
‘‘N - ddof‘‘ is used instead. In standard statistical practice, ‘‘ddof=1‘‘
provides an unbiased estimator of the variance of the infinite population.
‘‘ddof=0‘‘ provides a maximum likelihood estimate of the variance for
normally distributed variables. The standard deviation computed in this
function is the square root of the estimated variance, so even with
‘‘ddof=1‘‘, it will not be an unbiased estimate of the standard deviation
per se.

Note that, for complex numbers, ‘std‘ takes the absolute
value before squaring, so that the result is always real and nonnegative.

For floating-point input, the *std* is computed using the same
precision the input has. Depending on the input data, this can cause
the results to be inaccurate, especially for float32 (see example below).
Specifying a higher-accuracy accumulator using the ‘dtype‘ keyword can
alleviate this issue.

Examples
--------
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
```

```
1.1180339887498949 # may vary
>>> np.std(a, axis=0)
array([1.,  1.])
>>> np.std(a, axis=1)
array([0.5,  0.5])

In single precision, std() can be inaccurate:

>>> a = np.zeros((2, 512*512), dtype=np.float32)
>>> a[0, :] = 1.0
>>> a[1, :] = 0.1
>>> np.std(a)
0.45000005

Computing the standard deviation in float64 is more accurate:

>>> np.std(a, dtype=np.float64)
0.44999999925494177 # may vary

Specifying a where argument:

>>> a = np.array([[14, 8, 11, 10], [7, 9, 10, 11], [10, 15, 5, 10]])
>>> np.std(a)
2.614064523559687 # may vary
>>> np.std(a, where=[[True], [True], [False]])
2.0
```