Exercise Set 12 - Solution

1 Skiing and Fondue [normal]

We run a 2-factor,2-level ANOVA. We denote skiing as the first and fondue as the second factor, and assign the 2 levels "0" and "1" to them. So, $x_{0,1,5}$ is the 5th person in the group that did not ski but had fondue. This person rated the day with 4, thus $x_{0,1,5} = 4$.

a) First we compute the group means:

$$\bar{x}_{0,0,\bullet} = 4.22, \quad \bar{x}_{0,1,\bullet} = 6.22, \quad \bar{x}_{1,0,\bullet} = 6.33, \quad \bar{x}_{1,1,\bullet} = 7.67$$

The partial (also known as marginal) means and the total mean are

$$\bar{x}_{0,\bullet,\bullet} = 5.22, \quad \bar{x}_{1,\bullet,\bullet} = 7.00, \quad \bar{x}_{\bullet,0,\bullet} = 5.28, \quad \bar{x}_{\bullet,1,\bullet} = 6.94, \quad \bar{x}_{\bullet,\bullet,\bullet} = \bar{x}_T = 6.11$$

We get a table of means:

.	no Skiing	Skiing	all
no Fondue	4.22	6.33	5.28
Fondue	6.22	7.67	6.94
all	5.22	7.00	6.11

b) Next we can calculate the sum squared errors within each group:

$$SS_{0,0} = 31.6, \quad SS_{0,1} = 25.6, \quad SS_{1,0} = 50.0, \quad SS_{1,1} = 20.0$$

This leads us to the sum of squares as:

$$SS_E = \sum_{i=1}^{2}\sum_{j=1}^{2} SS_{i,j} = 127.11$$

Note that if we had not been given the raw data, but only the unbiased estimator for the variance for each group and the number of elements in each group ($n_{i,j}$) we could have used:

$$SS_E = \sum_{i=1}^{2}\sum_{j=1}^{2}(n_{i,j} - 1)s_{i,j}^2$$

For the total SS, we need to go back to the raw data (the reference point is the total mean). We find:

$$SS_T = \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{9}(x_{i,j,k} - \bar{x}_T)^2 = 181.56$$

Looking at effects of each factor, and knowing that $I = J = 2$ and $n = 9$ for each group, we find:

$SS_{B,skiing} = 9 * 2 * ((\bar{x}_{0,\bullet,\bullet} - \bar{x}_{\bullet,\bullet,\bullet})^2 + (\bar{x}_{1,\bullet,\bullet} - \bar{x}_{\bullet,\bullet,\bullet})^2) = 28.44$
$SS_{B,fondue} = 9 * 2 * ((\bar{x}_{\bullet,0,\bullet} - \bar{x}_{\bullet,\bullet,\bullet})^2 + (\bar{x}_{\bullet,1,\bullet} - \bar{x}_{\bullet,\bullet,\bullet})^2) = 25$
$\nu_{skiing} = I - 1 = 2 - 1 = 1$ $\nu_{fondue} = J - 1 = 2 - 1 = 1$
$\nu_E = IJ(n-1) = 2 \times 2 < \times (10 - 1) = 32$

This leads us to the ANOVA table below.

Source	ν	SS	MS	F
Skiing	1	28.44	28.44	7.15
Fondue	1	25	25	6.29
Interaction - to be discussed				
Error	32	127.11	3.92	
Total	35	181.56		

Using $\alpha = 0.05$ we find a critical $\mathcal{F}_{1,32} = 4.149$. Both factors have the same degrees of freedom, so they have the same critical F-value. Both factors well exceed the critical F-value, so both have a statistically significant effect on peoples happiness. Interactions will be dealt with in the next exercise.

2 Exam-style Python questions [basic-normal]

a) `3`
Remember that the *first* item in a numpy (and generally Python) array has the index 0. So index 1 will actually give the second item in the sorted list.

b) `[13 12 17 15]`
When performing a mathematical operation on a numpy array, it is performed on every item separately.

c) `(4,)`
`(2, 2)`
`myArray` is a a 1-d array (with length 4), whereas `reshapedArray` is a 2x2 array.

d) It will give an error message ("ValueError: operands could not be broadcast together with shapes (4,) (5,)"). As adding two numpy arrays (say a and b) means adding the first element of a to the first element of b etc., the two arrays have to be the same length.

e) `myUnbiasedStd = np.std(myArray,ddof=1)` makes sure that we devide by $N - 1$.

f) The most straightforward (and fastest) way is to use the built-in minimum-finding function `np.min(myArray)`. For this particular code, as we have already sorted the numbers, we could also just look up the first number in the sorted array, using `sortedArray[0]`

g) - When adding up two lists, the second list is appended to the first list, whereas when adding two arrays, the values are added element by element.
- Multiplying a list $*2$ gives a list that is twice as long (for example `[1,2]*2` gives `[1,2,1,2]` but pultiplying a numpy array $*2$ multipiles each element times 2 and keeps the array size the same.
- Whereas numpy functions can be run on lists (they will be converted to arrays in the process), using attributes (such as `myArray.mean()` does no work for lists. Numpy arrays can be used in built-in mathematical functions (e.g. it is possible to write `myArray**2`), giving piece-wise operations. This does not work for lists. Note that something like `np.sin(myArray**2)` will also work on a list, as it gets converted to an array in the process.
- Generally (operations on) numpy arrays are also (faster) more memory efficient.

3 Second Exam question from 2024

The correction for the exam question is given in the following pages as was used to correct last year's exams.

2 TOTAL: 25 PTS

2 a) ① $P(\text{"odd"}) = \frac{3}{5} = 0.6 = P(A)$

$P(x>3) = \frac{2}{5} = 0.4 = P(B)$

---

2 b) ② Independent if $P(A \cap B) = P(A) \cdot P(B)$.

$P(A \cap B) = \frac{1}{5} = 0.2$ (only "5")

$P(A) \cdot P(B) = \frac{3}{5} \cdot \frac{2}{5} = \frac{6}{25} = 0.24$

$\Rightarrow$ Not equal $\Rightarrow$ Not independent.

ALTERNATIVE SOLUTIONS:

I) Independent if $P(A|B) = P(A)$

$P(A|B) = \frac{1}{2} \neq \frac{3}{5}$

$\Rightarrow$ not independent.

II) Independent if $P(B|A) = P(B)$

$P(B|A) = \frac{1}{2} \neq \frac{2}{5} \Rightarrow$ not independent

2 c) $\mathbb{E}(x) = \sum_i x_i p_i = \frac{1}{5} + \frac{2}{5} + \frac{3}{5} + \frac{4}{5} + \frac{5}{5}$

(1)

$= \frac{15}{5} = \underline{3} = \mu$

2 d) $\mathbb{E}(x^2 - 10) = \sum_i (x_i^2 - 10) p_i$

(2)

$= \frac{(1^2 + 2^2 + 3^2 + 4^2 + 5^2 - 5 \cdot 10)}{5}$

$= \frac{55 - 50}{5} = \underline{+1}$

gain of 1 per round.

0 pts if they took $E(x)^2 - 10 = \mu^2 - 10 = 9 - 10 = -1$

2 e) $\bar{x} = \frac{2+5+2+11}{5} = 3$ (mean) 0.5pt

(2) $\tilde{x}_{50\%} = 2$ 0.5pt (Median. ordered list: 1, 2, 2, 5, 5)

$$s^2 = \frac{1}{n_s - 1} \sum_i (x_i - \bar{x})^2$$

$$= \frac{1}{4}\left((-2)^2 + (-1)^2 \cdot 2 + (2)^2 \cdot 2\right)$$

$$= \frac{14}{4} = 3.5$$

1 pt (0.5 if s, not s^2 was computed)

2f) (1) Fair die; expect $N_s \cdot p_i = 200 \cdot \frac{1}{6} = \underline{40}$ for each i

Result:	1	2	3	4	5
N occurred:	48	35	28	33	46
N expected:	40	40	40	40	40
Difference:	+8	-5	-2	-7	+6

2g) (1) A χ^2 (chi-2, chi-squared) test.

(Because we compare absolute (or relative) frequencies to ~~the~~ expected values (or probabilities).

2 h) (4)

$$\chi^2 = \sum_i \frac{(n_i - N_S p_i)^2}{N_S p_i} = N_S \sum_i \frac{\left(\frac{n_i}{N_S} - p_i\right)^2}{p_i}$$

(Abs. freq. ← n_i; expected ← $N_S p_i$; rel. freq. ← $\frac{n_i}{N_S}$)

$$= \frac{|+8|^2 + |-5|^2 + |-2|^2 + |-7|^2 + |+6|^2}{200 \cdot \frac{1}{5}}$$

$$= \frac{178}{40} = \underline{4.45} \quad 1\text{pt}$$

Degrees of freedom: $df = \nu = K - 1 = 5 - 1 = \underline{4}$ (← Number of possible outcomes) 1pt

$$q\chi^2_{\nu=4} = 9.488 \quad 0.5\text{pt}$$

$$\chi^2 = 4.45 < 9.488 = q\chi^2_{\nu=4}$$

1.5pt → χ^2 is small enough that at the 95% confidence level we can say the die is fair.
(we cannot reject the null hypothesis "the die is fair")
(we cannot say the die is unfair).

2 i) We need to consider the expected absolute frequency for the least probable result.

(1) $N_S \cdot \min(p_i)$. Here all $p_i = \frac{1}{5}$, so

$N_S \cdot \min(p_i) = N_S \cdot p_i = \frac{200}{5} = 40$.

This has to be large enough, and typically $N_S \cdot \min(p_i) \geq 5$ is considered sufficient.

No points for general statements.

Must mention expected value (0.5) of least likely outcome (0.5²).

2 j) Sorted Array:

(3) [1 2 2 5 5]

Length of array:

5

Position of the median:

3

The median:

5

output

output

output

0.5 pt

0.5 pt

1 pt

) no subtraction for wrong syntax (C) or ...)

2 k) (2) median = sortedArray[3] which is the 4th element in the array - Python starts counting from 0. } 1pt

1pt { Possible solutions: I) median = sortedArray[medianPosition-1]

II) medianPosition Float = arrayLength/2 - 0.5

III) other similar solutions

Just saying myMedian = np.median(myArray) gives only 0.5 pt.

2 e) possible solutions: I) my Median = np.median(myArray)

(3)

II): Use an if-clause to check if arrayLength is even or odd. (e.g. by arrayLength % 2 == 0)

1 { if odd → code as above (with correction)

0.5 { if even → take

1.5 { (sorterArray[arrayLength/2] + sorterArray[arrayLength-1])/2

---

2 m) Proof by counterexample: {-2, -2, 0, 1, 3}

(1) has mean = 0 and median = 0 but it is not symmetric around 0.

0.5pt for answer w/o proof.

1pt for "handwavy" proof.