Homework 1

Problem 1 Statistical Properties

(a) Max and min:

```python
# 1(a) Max and Min
# Max
def getMax(array):
    max = 0
    for item in array:
        if item > max:
            max = item
    # return np.max(array)
    return max

Midterm_Max = getMax(Midterm)
print("Midterm Max:", Midterm_Max)
Final_Max = getMax(Final)
print("Final Max:", Final_Max)

# Min
def getMin(array):
    min = 1000
    for item in array:
        if item < min:
            min = item
    # return np.min(array)
    return min

Midterm_Min = getMin(Midterm)
print("Midterm Min:", Midterm_Min)
Final_Min = getMin(Final)
print("Final Min:", Final_Min)
```

```
Midterm Max: 99
Final Max: 100
Midterm Min: 75
Final Min: 77
```

- Max:
    - Midterm = 99
    - Final = 100
- Minx:
    - Midterm = 75
    - Final = 77

(b) Mean, mode and median:

```python
# Mean
def getMean(array):
    sum = 0
    for item in array:
        sum = sum + item
    amount = len(array)
    mean = sum / amount
    return mean

Midterm_Mean = getMean(Midterm)
print("Midterm Mean:", Midterm_Mean)
Final_Mean = getMean(Final)
print("Fianl Mean:", Final_Mean)

# Mode
def getMode(array):
    counts = np.bincount(array)     # get frequncy
    max_count = getMax(counts)      # get highest frequency

    return np.where(counts==max_count)

Midterm_Mode = getMode(Midterm)
print("Midterm Mode:", Midterm_Mode)
Final_Mode = getMode(Final)
print("Final Mode:", Final_Mode)

# Median
def getMedian(array):
    return np.median(array)

Midterm_Median = getMedian(Midterm)
print("Midterm Median:", Midterm_Median)
Final_Median = getMedian(Final)
print("Final Median:", Final_Median)
```

```
Midterm Mean: 88.7
Fianl Mean: 88.2
Midterm Mode: (array([96]),)
Final Mode: (array([80, 88]),)
Midterm Median: 89.0
Final Median: 88.0
```

- Mean:
    - Midterm: 88.7
    - Final: 88.2
- Mode:
    - Midterm: 96
    - Final: 80, 88
- Median:
    - Midterm: 89
    - Final: 88

(c) First quartile, third quartile and inter-quartile range:

```python
# 1(c)First Quartile, Third Quartile and Inter-quartile Range
# First Quartile
def getFirstQuartile(array):
    array = np.sort(array)    # sort array
    length = len(array)
    mid = (int)(length / 2)
    first = (int)(mid / 2)
    third = mid + first

    if length >= 4:
        if mid % 2== 0:
            first_quartile = (array[first-1] + array[first]) / 2
            third_quartile = (array[third-1] + array[third+1]) / 2
        else:
            first_quartile = array[first]
            third_quartile = array[third]

    inter_quartile = third_quartile - first_quartile

    return array, first_quartile, third_quartile, inter_quartile

[Midterm_Sorted, Midterm_First_Quartile, Midterm_Third_Quartile, Mid
print("Midterm Sorted Array:", Midterm_Sorted)
print("Midterm First Quartile:", Midterm_First_Quartile)
print("Midterm Third Quartile:", Midterm_Third_Quartile)
print("Midterm Inter Quartile:", Midterm_Inter_Quartile)
[Final_Sorted, Final_First_Quartile, Final_Third_Quartile, Final_Int
print("Final Sorted Array:", Final_Sorted)
print("Final First Quartile:",Final_First_Quartile)
print("Final Third Quartile:", Final_Third_Quartile)
print("Final Inter Quartile:", Final_Inter_Quartile)
```

```
Midterm Sorted Array: [75 78 84 86 88 90 95 96 96 99]
Midterm First Quartile: 84
Midterm Third Quartile: 96
Midterm Inter Quartile: 12
Final Sorted Array: [ 77  80  80  85  88  88  90  95  99 100]
Final First Quartile: 80
Final Third Quartile: 95
Final Inter Quartile: 15
```

- Midterm:

75	78	84	86	88	90	95	96	96	99

  - First Quartile: 84
  - Third Quartile: 96
  - Inter-Quartile: 12
- Final:

77	80	80	85	88	88	90	95	99	100

  -
    - First Quartile: 80
    - Third Quartile: 95
    - Inter-Quartile: 15

(d) Variance(sample & population):

```python
# 1(d)Variance (Sample & Population)
def getVariancePopulation(array):
    length = len(array)
    square_sum = 0
    mean = getMean(array)

    for item in array:
        square_sum = square_sum + (item - mean) * (item - mean)

    population_var = round(square_sum / length, 3)

    # return np.var(array)
    return population_var

def getVarianceSample(array):
    length = len(array)
    square_sum = 0
    mean = getMean(array)

    for item in array:
        square_sum = square_sum + (item - mean) * (item - mean)

    sample_var = round(square_sum / (length - 1) , 3)

    # return np.var(array)
    return sample_var

Midterm_Population_Variance = getVariancePopulation(Midterm)
print("Midterm Population Variance:", Midterm_Population_Variance)
Midterm_Sample_Variance = getVarianceSample(Midterm)
print("Midterm Sample Variance:", Midterm_Sample_Variance)
Final_Population_Variance = getVariancePopulation(Final)
Final_Sample_Variance = getVarianceSample(Final)
print("Final Population Variance:", Final_Population_Variance)
print("Final Sample Variance:", Final_Sample_Variance)
```

```
Midterm Population Variance: 58.61
Midterm Sample Variance: 65.122
Final Population Variance: 57.56
Final Sample Variance: 63.956
```

- Population:

$$
Var = E((X-\overline{x})^2) = E(x^2) - [E(x)]^2
$$

$$
Var = \frac{1}{n}*\sum_{i=1}^{n}(x_i - \overline{x})^2
$$

- Sample:

$
Var = \frac{1}{n-1}*\sum_{i=1}^{n}(x_i - \overline{x})^2
$

- Midterm:
  - Population Variance: 58.610
  - Sample Variance: 65.122
- Final:
  - Population Variance: 57.560
  - Sample Variance: 63.956

(e) Standard Deviation (sample & population):

```
# 1(e)Standard Deviation
def getStandardDeviationPopulation(array):
    # return np.std(array)
    return round(np.sqrt(getVariancePopulation(array)),3)

def getStandardDeviationSample(array):
    # return np.std(array)
    return round(np.sqrt(getVarianceSample(array)), 3)

Midterm_Standard_Population_Deviation = getStandardDeviationPopulati
print("Midterm Standard Population Deviation:", Midterm_Standard_Pop
Midterm_Standard_Sample_Deviation = getStandardDeviationSample(Midte
print("Midterm Standard Sample Deviation:", Midterm_Standard_Sample_
Final_Standard_Population_Deviation = getStandardDeviationPopulation
print("Final Standard Population Deviation:", Final_Standard_Populat
Final_Standard_Sample_Deviation = getStandardDeviationSample(Final)
print("Final Standard Sample Deviation:", Final_Standard_Sample_Devi
```

```
Midterm Standard Population Deviation: 7.656
Midterm Standard Sample Deviation: 8.07
Final Standard Population Deviation: 7.587
Final Standard Sample Deviation: 7.997
```

- Standard Deviation:

  $
  Std = \sqrt{Var}
  $

- Midterm:
  - Population Standard Deviation: 7.656
  - Sample Standard Deviation: 8.070
- Final:
  - Population Standard Deviation: 7.587
  - Sample Standard Deviation: 7.997

Problem 2 Normalize Data

(a) Min-Max Normalization:

```python
# 2(a) Min-Max Normalization
def getMinMaxNormalization(array):
    old_min = getMin(array)
    old_max = getMax(array)
    new_min = 0
    new_max = 1

    new_list = []
    for item in array:
        new_list.append(round((item-old_min)/(old_max-old_min)*(new_

    new_array = np.array(new_list)
    return new_array

Midterm_MinMax_Normalization = getMinMaxNormalization(Midterm)

print("Midterm MinMax Normalization Student 1:", Midterm_MinMax_Norm
print("Midterm MinMax Normalization Student 2:", Midterm_MinMax_Norm
print("Midterm MinMax Normalization Student 3:", Midterm_MinMax_Norm
```

```
Midterm MinMax Normalization Student 1: 0.833
Midterm MinMax Normalization Student 2: 0.458
Midterm MinMax Normalization Student 3: 0.125
```

- $$v' = \frac{v -min}{max - min}*(newMax - newMin) + newMin$$
- Student 1: 0.833
- Student 2: 0.458
- Student 3: 0.125

(b) Variance(population) of min-max normalized midterm scored:

```python
# 2(b) Variance(Population) of Min-Max Normalization
def getVarianceMinMaxNormalization(array):
    new_array = getMinMaxNormalization(array)
    return getVariancePopulation(new_array)

Midterm_Variance_MinMax_Normalization = getVarianceMinMaxNormalizati
print("Midterm Variance MinMax Normalization:", Midterm_Variance_Min
```

```
Midterm Variance MinMax Normalization: 0.102
```

- Variance (Population):

  $$Var = \frac{1}{n}*\sum_{i=1}^{n}(x_i - \overline{x})^2$$
- Min-Max Normalization:

  $$v' = \frac{v -min}{max - min}*(newMax - newMin) + newMin$$
- Midterm Variance Min-Max Normalization:0.102

(c) Z-Score Normalization of final scores:

```python
# 2(c) Z-Score Normalizaion
def getZScoreNormalization(array):
    mean = getMean(array)
    std = getStandardDeviationPopulation(array)

    new_list = []
    for item in array:
        new_list.append(round((item-mean)/std,3))

    new_array = np.array(new_list)
    return new_array

Final_ZScore_Normalization = getZScoreNormalization(Final)
print("Final ZScore Normalization Student 4:", Final_ZScore_Normaliz
print("Final ZScore Normalization Student 5:", Final_ZScore_Normaliz
print("Final ZScore Normalization Student 6:", Final_ZScore_Normaliz
```

```
Final ZScore Normalization Student 4: 0.896
Final ZScore Normalization Student 5: -0.422
Final ZScore Normalization Student 6: -1.476
```

- $v' = \frac{v - \mu}{\sigma}$
- Student 4: 0.896
- Student 5: -0.422
- Student 6: -1.476

(d) Variance(population) of z-score normalized final score:

```python
# 2(d) Variance(Population) of Z-Score Normalization
def getVarianceZScoreNormalization(array):
    new_array = getZScoreNormalization(array)
    return getVariancePopulation(new_array)

Final_Variance_ZScore_Normalization = getVarianceZScoreNormalization
print("Final Variance ZScore Normalization:", Final_Variance_ZScore_
```

```
Final Variance ZScore Normalization: 1.0
```

- Variance (Population):

  $Var = \frac{1}{n}*\sum_{i=1}^{n}(x_i - \overline{x})^2$

- Z-Score Normalization:

  $v' = \frac{v - \mu}{\sigma}$

- Variance(population) of z-score normalized final score: 1.000

Problem 3 Relationship between Midterm and Final

(a) Covariance(Population):

```python
# 3(a) Covariance(population)
def getCovariancePopulation(array1, array2):
    mean1 = getMean(array1)
    mean2 = getMean(array2)
    length = len(array1)

    covariance = sum(array1 * array2) / length - mean1 * mean2
    return round(covariance, 3)

Covariance = getCovariancePopulation(Midterm, Final)
print("Covariance:", Covariance)
```

```
Covariance: 18.16
```

$$
Cov = E[(x_1-\mu_1)*(x_2=\mu_2)]=E(x_1*x_2)-\mu_1*\mu_2
$$

- Covariance: 18.160

(b)Pearson's Correlation Coefficient(population standard deviation and covariance):

```python
# 3(b) Pearson's Correlation Coefficient
def getPearsonCorrelation(array1, array2):
    covariance = getCovariancePopulation(array1, array2)
    std1 = getStandardDeviationPopulation(array1)
    std2 = getStandardDeviationPopulation(array2)

    pearson = covariance / (std1 * std2)
    return round(pearson, 3)

PearsonCorrelation = getPearsonCorrelation(Midterm, Final)
print("Peasrson Correlation Coefficient:", PearsonCorrelation)
```

```
Peasrson Correlation Coefficient: 0.313
```

- $$
  Pearson = \frac{cov(x_1, x_2)}{\sigma_{x_1} * \sigma_{x_2}}
  $$

  Pearson Correlation Coefficient: 0.313

(c) Independent?

- Since the Pearson Correlation Coefficient is 0.313, which means M which denotes midterm scores and F denotes final scores are not independent and are positive related.

(d) Distance

```python
# 3(d) Distance
# Manhattan Distance
def getManhattan(array1, array2):
    length = len(array1)

    sum = 0
    for iter in range(length):
        sum = sum + abs(array1[iter] - array2[iter])

    return sum

Manhattan = getManhattan(Midterm, Final)
print("Manhattan Distance:", Manhattan)

# Euclidean Distance
def getEuclidean(array1, array2):
    length = len(array1)

    sum = 0
    for iter in range(length):
        sum = sum + pow((array1[iter] - array2[iter]),2)

    return round(pow(sum, 0.5), 3)

Euclidean = getEuclidean(Midterm, Final)
print("Euclidean Distance:", Euclidean)
```

```python
# Supremum Distance
def getSupremum(array1, array2):
    length = len(array1)

    max = 0
    for iter in range(length):
        temp = abs(array1[iter] - array2[iter])
        if max < temp:
            max = temp

    return round(temp, 3)

Supremum = getSupremum(Midterm, Final)
print("Supremum Distance:", Supremum)

# Cosine Similarity
def getCosineSimilarity(array1, array2):
    cosine = np.dot(array1, array2) / (pow(sum(pow(array1, 2)), 0.5)

    #return np.dot(array1, array2) / (norm(array1)*norm(array2))
    return round(cosine, 3)

Cosine = getCosineSimilarity(Midterm, Final)
print("Cosine Similarity:", Cosine)
```

```
Manhattan Distance: 75
Euclidean Distance: 28.302
Supremum Distance: 16
Cosine Similarity: 0.995
```

- Minkowski Distance:

  $
  d_{ij} = \sqrt[p]{|x_{i1}-x_{j1}|^{p}+|x_{i2}-x_{j2}|^{p}+L+|x_{il}-x_{jl}|^{p}}
  $

- Manhattan Distance:
  -

  $
  d_{ij} = |x_{i1}-x_{j1}|+|x_{i2}-x_{j2}|+L+|x_{il}-x_{jl}|
  $
  - Midterm & Final: 75
- Euclidean Distance:
  -

  $
  d_{ij} = \sqrt[2]{|x_{i1}-x_{j1}|^{2}+|x_{i2}-x_{j2}|^{2}+L+|x_{il}-x_{jl}|^{2}}
  $
  - Midterm & Final: 28.302
- Supremum Distance:
  -

$$
d_{ij} = lim_{p\rightarrow \infty}\sqrt[p]{|x_{i1}-x_{j1}|^{p}+|x_{i2}-x_{j2}|^{p}+L+|x_{il}-x_{jl}|^{p}} = max_{f=1}^{l}|x_{if} - x_{jf}|
$$

  - Midterm & Final: 16
- Cosine Similarity of m and f:
  -

$$
CosineSimilarity = \frac{A\cdot B}{||A|| * ||B||}= \frac{\sum_{i=1}^{n}A_i * B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}*\sqrt{\sum_{i=1}^{n}(B_i)^2}
$$

  - Midterm & Final: 0.995

(e)

Manhattan Distance: Means the difference of each student between midterm and final. The distance is always positive and the larger value means the difference is larger. In the scenario, the larger value means students perform different in midterm and final.

(f) Kullback-Leibler Divergence / Jaccard Coefficient

I don't think using Kullback-Leibler Divergence is a good choice since the model is discrete and some of the p(x) will be zero which will lead mistake when taking log. Besides, I also don't think using Kullback-Leibler Divergence is a good choice since the attribute is binary while it's not appropriate to separate Midterm and Final in this way.

- KL:

$$
D_{kL}(p(x)||q(x))=\sum q(x) * ln\frac{p(x)}{q(x)}
$$

- Jaccard:

$$
Jaccard= \frac{|a\cap b|}{|A| + |B| - |A \cap B|)}
$$

Problem 4

(a) X^2 correlation value:

$$
X^2=\sum_{i=1}^{n}\frac{(O_i-E_i)^2}{E_i}
$$

```python
# 4(a) x^2 correlation
def getX2Correlation(array):
    length = array.shape[0]

    sum_row_0 = array[0][0] + array[0][1]
    sum_row_1 = array[1][0] + array[1][1]
    sum_col_0 = array[0][0] + array[1][0]
    sum_col_1 = array[0][1] + array[1][1]
    sum_total = sum_row_0 + sum_row_1

    new_array = np.zeros([2,2])
    new_array[0][0] = sum_row_0 * sum_col_0 / sum_total
    new_array[0][1] = sum_row_0 * sum_col_1 / sum_total
    new_array[1][0] = sum_row_1 * sum_col_0 / sum_total
    new_array[1][1] = sum_row_1 * sum_col_1 / sum_total

    sum = 0
    for i in range(length):
        for j in range(length):
            sum = sum + pow((array[i][j] - new_array[i][j]), 2) / ne
    return round(sum, 3)

X2Correlation = getX2Correlation(Purchase)
print("X2 Correlation:", X2Correlation)
```

```
X2 Correlation: 2062.333
```

(b)

```
We can reject the null hypothesis of independence at a confidence level of 0.001. Therefore, "purchasing beer" and "purchasing diaper" are correlated.
```

(c)

```
p = [200 : 80+20 : 3000] = [2 : 1 : 30] = [0.061 : 0.030 : 0.909]
```

(d) Kullback-Leibler Divergence

```python
# 4(d) Kullback-Leibler Divergence
def getKL(p, q):
    length = len(p)
    sum = 0
    for iter in range(length):
        sum = sum + q[iter] *np.log(q[iter]/p[iter])

    return round(sum, 3)

p = np.array([0.061, 0.030, 0.909])
q = np.array([0.5, 0.3, 0.2])
KL = getKL(p, q)
print("Kullback Leibler Divergence:", KL)
```

```
Kullback Leibler Divergence: 1.44
```

$
D_{kL}(p(x)||q(x))=\sum q(x) * ln\frac{p(x)}{q(x)}
$

p = [0.061 0.030 0.909]

```
q = [0.5 0.3 0.2 ]

KL = 1.440
```