Multivariate_Analysis_Assignment_4

March 14, 2021

```
In [1]: # Problems on the Analysis of Variance
```

0.1 *Problem* 1

```
In [2]: import numpy as np
        import scipy as sc
        from scipy.stats import f

In [3]: data = [[551, 595, 639, 417, 563],
                [457, 580, 615, 449, 631],
                [450, 508, 511, 517, 522],
                [731, 583, 573, 438, 613],
                [499, 633, 648, 415, 656],
                [632, 517, 677, 555, 679]]

        data = np.asmatrix(data)
        k = data.shape[1]
        n = data.shape[0]

        print('k = ' + str(k))
        print('n = ' + str(n))

k = 5
n = 6

In [4]: total_mean = np.mean(data)
        row_means = np.asarray(np.ravel(np.mean(data, axis = 0)))

        print('Total data mean = ' + str(total_mean))
        print('Row means = ' + str(np.round(row_means, 4)))

Total data mean = 561.8
Row means = [553.3333 569.3333 610.5    465.1667 610.6667]

In [5]: numerator = (n / (k - 1)) * np.matmul((row_means - total_mean),
                                              (row_means - total_mean))
```

```
    denominator = 0

    for j in range(n):
      for i in range(k):
        denominator += np.power((data[j, i] - row_means[i]), 2)

    denominator /= (k * (n - 1))

    F_value = numerator / denominator
    alpha = 0.05
    F_critical = f.isf(alpha, k - 1, k * (n - 1))

    print('Computed F Value = ' + str(F_value))
    print('Critical F Value = ' + str(F_critical))
    print('Computed F value > Critical F value = ' + str(F_value > F_critical))
```

```
Computed F Value = 4.30153590405874
Critical F Value = 2.758710469717632
Computed F value > Critical F value = True
```

$ We reject H_0.$

0.2 *Problem* 2

```
In [6]: set_1 = np.asmatrix([[1.11,       2.569,       3.58,        0.760],
                             [1.19,       2.928,       3.75,        0.821],
                             [1.09,       2.865,       3.93,        0.928],
                             [1.25,       3.844,       3.94,        1.009],
                             [1.11,       3.027,       3.60,        0.766],
                             [1.08,       2.336,       3.51,        0.726],
                             [1.11,       3.211,       3.98,        1.209],
                             [1.16,       3.037,       3.62,        0.750]])

        set_2 = np.asmatrix([[1.05,       2.074,       4.09,        1.036],
                             [1.17,       2.885,       4.06,        1.094],
                             [1.11,       3.378,       4.87,        1.635],
                             [1.25,       3.906,       4.98,        1.517],
                             [1.17,       2.782,       4.38,        1.197],
                             [1.15,       3.018,       4.65,        1.244],
                             [1.17,       3.383,       4.69,        1.495],
                             [1.19,       3.447,       4.40,        1.026]])

        set_3 = np.asmatrix([[1.07,       2.505,       3.76,        0.912],
                             [0.99,       2.315,       4.44,        1.398],
                             [1.06,       2.667,       4.38,        1.197],
                             [1.02,       2.390,       4.67,        1.613],
                             [1.15,       3.021,       4.48,        1.476],
```

```python
                    [1.20,        3.085,        4.78,        1.571],
                    [1.20,        3.308,        4.57,        1.506],
                    [1.17,        3.231,        4.56,        1.458]])


set_4 = np.asmatrix([[1.22,        2.838,        3.89,        0.944],
                    [1.03,        2.351,        4.05,        1.241],
                    [1.14,        3.001,        4.05,        1.023],
                    [1.01,        2.439,        3.92,        1.067],
                    [0.99,        2.199,        3.27,        0.693],
                    [1.11,        3.318,        3.95,        1.085],
                    [1.20,        3.601,        4.27,        1.242],
                    [1.08,        3.291,        3.85,        1.017]])

set_5 = np.asmatrix([[0.91,        1.532,        4.04,        1.084],
                    [1.15,        2.552,        4.16,        1.151],
                    [1.14,        3.083,        4.79,        1.381],
                    [1.05,        2.330,        4.42,        1.242],
                    [0.99,        2.079,        3.47,        0.673],
                    [1.22,        3.366,        4.41,        1.137],
                    [1.05,        2.416,        4.64,        1.455],
                    [1.13,        3.100,        4.57,        1.325]])

set_6 = np.asmatrix([[1.11,        2.813,        3.76,        0.800],
                    [0.75,        0.840,        3.14,        0.606],
                    [1.05,        2.199,        3.75,        0.790],
                    [1.02,        2.132,        3.99,        0.853],
                    [1.05,        1.949,        3.34,        0.610],
                    [1.07,        2.251,        3.21,        0.562],
                    [1.13,        3.064,        3.63,        0.707],
                    [1.11,        2.469,        3.95,        0.952]])

data = [set_1, set_2, set_3, set_4, set_5, set_6]
total = np.vstack([set_1, set_2, set_3, set_4, set_5, set_6])

p = np.shape(set_1)[1]
n = np.shape(set_1)[0]
k = len(data)
N = n * k

print('p = ' + str(p))
print('n = ' + str(n))
print('k = ' + str(k))
print('N = ' + str(N))
```

```
p = 4
n = 8
k = 6
```

```
N = 48
```

0.2.1 *(a)*

```
In [7]: means = []
        print('Unit means: ')
        print()

        for index in range(len(data)):
          means.append(np.mean(data[index], axis = 0))
          print('mean[i = ' + str(index + 1) + '] = ' + str(np.ravel(means[-1])))

        mean_all = np.mean(total, axis = 0)

        print()
        print('Total mean: ' + str(np.ravel(np.round(mean_all, 4))))
```

```
Unit means:

mean[i = 1] = [1.1375   2.977125 3.73875  0.871125]
mean[i = 2] = [1.1575   3.109125 4.515    1.2805  ]
mean[i = 3] = [1.1075   2.81525  4.455    1.391375]
mean[i = 4] = [1.0975  2.87975 3.90625 1.039  ]
mean[i = 5] = [1.08    2.55725 4.3125  1.181  ]
mean[i = 6] = [1.03625  2.214625 3.59625  0.735   ]

Total mean: [1.1027 2.7589 4.0873 1.083 ]
```

```
In [8]: H = 0
        for i in range(k):
          H += np.matmul((means[i] - mean_all).T, (means[i] - mean_all))

        H = n * H
        print('H = ')
        print()
        print(np.round(H, 4))
```

```
H =

[[0.0736 0.5374 0.3323 0.2085]
 [0.5374 4.1997 2.3554 1.6371]
 [0.3323 2.3554 6.1139 3.781 ]
 [0.2085 1.6371 3.781  2.4931]]
```

```
In [9]: E = 0
        for i in range(k):
```

```
        for j in range(n):
            unit = data[i][j] - means[i]
            E += np.matmul(unit.T, unit)

    print('E = ')
    print()
    print(np.round(E, 4))
```

```
E =

[[ 0.32    1.6966  0.5541  0.2171]
 [ 1.6966 12.1428  4.3636  2.1102]
 [ 0.5541  4.3636  4.2908  2.4817]
 [ 0.2171  2.1102  2.4817  1.7225]]
```

```
In [10]: ratio = np.linalg.det(E) / np.linalg.det(E + H)
         nu_H = k - 1
         nu_E = k * (n - 1)

         print('p = ' + str(p))
         print('nu_H = ' + str(nu_H))
         print('nu_E = ' + str(nu_E))
```

```
p = 4
nu_H = 5
nu_E = 42
```

```
In [11]: w = nu_E + nu_H - (p + nu_H + 1) / 2
         t = np.sqrt((p * p * nu_H * nu_H - 4) / (p * p + nu_H * nu_H - 5))
         df_1 = float(p * nu_H)
         df_2 = w * t - (p * nu_H - 2) / 2

         print('w = ' + str(w))
         print('t = ' + str(t))
         print('df_1 = ' + str(df_1))
         print('df_2 = ' + str(df_2))
```

```
w = 42.0
t = 3.3166247903554
df_1 = 20.0
df_2 = 130.2982411949268
```

```
In [12]: step = np.power(ratio, (1 / t))
         F_calculated = ((1 - step) / step) * (df_2 / df_1)
         alpha = 0.05
         F_compare = f.isf(alpha, df_1, df_2)
```

```
In [13]: print('F_calculated = ' + str(F_calculated))
         print('F_compare = ' + str(F_compare))
         print('F_calculated > F_compare = ' + str(F_calculated > F_compare))
```

```
F_calculated = 4.936888039729533
F_compare = 1.6516770323178185
F_calculated > F_compare = True
```

We reject H_0.

0.2.2 (*b*)

```
In [14]: def T_square_value(alpha, p, v):
           # 'v' being the number of degrees of freedom
           # 'p' - variable setting

           return f.isf(alpha, p, v - p + 1) / (v - p + 1) * (v * p)
```

```
In [15]: C_1 = [2, -1, -1, -1, -1, 2]
         delta_1 = 0

         for index in range(len(C_1)):
           delta_1 += C_1[index] * means[index]

         sum_C1_square = np.matmul(np.asmatrix(C_1), np.asmatrix(C_1).T)

         T_1_squared = (n / sum_C1_square) * np.matmul(
             np.matmul(delta_1, np.linalg.inv(E / nu_E)), delta_1.T)

         T_1_squared = np.ravel(T_1_squared)[0]

         alpha = 0.05 / 2
         # Required Adjustment

         T_1_square_comp = T_square_value(alpha, p, nu_E)

         print('delta_1 = ' + str(np.round(np.ravel(delta_1), 4)))
         print('T_1_squared = ' + str(T_1_squared))
         print('T_1_square_comp = ' + str(T_1_square_comp))
         print('Calculated T square > T square value for comparison: '
               + str(T_1_squared > T_1_square_comp))
```

```
delta_1 = [-0.095  -0.9779 -2.5188 -1.6796]
T_1_squared = 52.74890659644772
T_1_square_comp = 13.506530727954198
Calculated T square > T square value for comparison: True
```

We reject H_0 for C_1.

```
In [16]: C_2 = [1, 0, 0, 0, 0, -1]
         delta_2 = 0

         for index in range(len(C_2)):
           delta_2 += C_2[index] * means[index]

         sum_C2_square = np.matmul(np.asmatrix(C_2), np.asmatrix(C_2).T)

         T_2_squared = (n / sum_C2_square) * np.matmul(
             np.matmul(delta_2, np.linalg.inv(E / nu_E)), delta_2.T)

         T_2_squared = np.ravel(T_2_squared)[0]

         alpha = 0.05 / 2
         # Required Adjustment

         T_2_square_comp = T_square_value(alpha, p, nu_E)

         print('delta_2 = ' + str(np.round(np.ravel(delta_2), 4)))
         print('T_2_squared = ' + str(T_2_squared))
         print('T_2_square_comp = ' + str(T_2_square_comp))
         print('Calculated T square > T square value for comparison: '
               + str(T_2_squared > T_2_square_comp))
```

```
delta_2 = [0.1012 0.7625 0.1425 0.1361]
T_2_squared = 13.972735901764274
T_2_square_comp = 13.506530727954198
Calculated T square > T square value for comparison: True
```

We reject H_0 for C_2.

0.2.3 (*c*)

```
In [17]: # Testing Individual Variables

         alpha = 0.05
         n = 8
         k = 6

         for variable in range(4):
           print('Testing for y' + str(variable + 1) + ":")
           print()
           data = np.vstack([np.ravel(set_1[:, variable].T),
                             np.ravel(set_2[:, variable].T),
                             np.ravel(set_3[:, variable].T),
                             np.ravel(set_4[:, variable].T),
```

```
                   np.ravel(set_5[:, variable].T),
                   np.ravel(set_6[:, variable].T)])

    data = data.T

    assert n == data.shape[0]
    assert k == data.shape[1]

    total_mean = np.mean(data)
    row_means = np.asarray(np.ravel(np.mean(data, axis = 0)))

    print('Total data mean = ' + str(total_mean))
    print('Row means = ' + str(np.round(row_means, 4)))

    numerator = (n / (k - 1)) * np.matmul(
        (row_means - total_mean), (row_means - total_mean))

    denominator = 0

    for j in range(n):
      for i in range(k):
        denominator += np.power((data[j, i] - row_means[i]), 2)

    denominator /= (k * (n - 1))

    F_value = numerator / denominator
    F_critical = f.isf(alpha, k - 1, k * (n - 1))

    print('Computed F Value = ' + str(F_value))
    print('Critical F Value = ' + str(F_critical))
    print('Computed F value > Critical F value = ' + str(F_value > F_critical))

    print()
    if F_value > F_critical:
      print('We reject H_0 for y' + str(variable + 1) + '.')
    else:
      print('We accept H_0 for y' + str(variable + 1) + '.')
    print()
```

```
Testing for y1:

Total data mean = 1.1027083333333332
Row means = [1.1375 1.1575 1.1075 1.0975 1.08   1.0363]
Computed F Value = 1.931036368608147
Critical F Value = 2.437692640311651
Computed F value > Critical F value = False

We accept H_0 for y1.
```

```
Testing for y2:

Total data mean = 2.758854166666667
Row means = [2.9771 3.1091 2.8152 2.8797 2.5572 2.2146]
Computed F Value = 2.9051938814597604
Critical F Value = 2.437692640311651
Computed F value > Critical F value = True

We reject H_0 for y2.

Testing for y3:

Total data mean = 4.087291666666666
Row means = [3.7387 4.515  4.455  3.9062 4.3125 3.5962]
Computed F Value = 11.969075204288243
Critical F Value = 2.437692640311651
Computed F value > Critical F value = True

We reject H_0 for y3.

Testing for y4:

Total data mean = 1.083
Row means = [0.8711 1.2805 1.3914 1.039  1.181  0.735 ]
Computed F Value = 12.157715876070867
Critical F Value = 2.437692640311651
Computed F value > Critical F value = True

We reject H_0 for y4.
```

Decisions :

- *Accept* H_0 *for* $y1$.
- *Reject* H_0 *for* $y2$.
- *Reject* H_0 *for* $y3$.
- *Reject* H_0 *for* $y4$.

0.3 *Problem* 3

In [18]:
```
import json

data_a = [[0.71, 2.20, 2.25],
          [1.66, 2.93, 3.93],
```

```
                 [2.01, 3.08, 5.08],
                 [2.16, 3.49, 5.82],
                 [2.42, 4.11, 5.84],
                 [2.42, 4.95, 6.89],
                 [2.56, 5.16, 8.50],
                 [2.60, 5.54, 8.56],
                 [3.31, 5.68, 9.44],
                 [3.64, 6.25, 10.52],
                 [3.74, 7.25, 13.46],
                 [3.74, 7.90, 13.57],
                 [4.39, 8.85, 14.76],
                 [4.50, 11.96, 16.41],
                 [5.07, 15.54, 16.96],
                 [5.26, 15.89, 17.56],
                 [8.15, 18.3, 22.82],
                 [8.24, 18.59, 29.13]]

        data_b = [[2.20, 4.04, 2.71],
                 [2.69, 4.16, 5.43],
                 [3.54, 4.42, 6.38],
                 [3.75, 4.93, 6.38],
                 [3.83, 5.49, 8.32],
                 [4.08, 5.77, 9.04],
                 [4.27, 5.86, 9.56],
                 [4.53, 6.28, 10.01],
                 [5.32, 6.97, 10.08],
                 [6.18, 7.06, 10.62],
                 [6.22, 7.78, 13.80],
                 [6.33, 9.23, 15.99],
                 [6.97, 9.34, 17.90],
                 [6.97, 9.91, 18.25],
                 [7.52, 13.46, 19.32],
                 [8.36, 18.4, 19.87],
                 [11.65, 23.89, 21.60],
                 [12.45, 26.39, 22.25]]

        data_a = np.asmatrix(data_a)
        data_b = np.asmatrix(data_b)

        # Using a natural logarithm transformation
        data_a = np.log(data_a)
        data_b = np.log(data_b)

        all_data = [data_a, data_b]
```

2 *types of iron used.* 3 *molarities for each type.* 18 *observations per cell.*

```
In [19]: mean_total = np.mean(np.vstack(all_data))
```

```
# i, j, k:
# i = 1 or 2 (types of Iron)
# j = 1, 2, or 3 (types of molarities)
# k = 1, 2, ... , 18 (observation index)

a = 2
b = 3
n = 18
```

In [20]:
```
mean_1__ = np.mean(data_a)
mean_2__ = np.mean(data_b)

mean_i__ = np.asarray([mean_1__, mean_2__])

print('Mean of A Chunks: ')
print(mean_i__)
print()

mean__j_ = np.ravel(np.mean(np.vstack([data_a, data_b]), axis = 0))

print('Mean of B Chunks: ')
print(mean__j_)
```

```
Mean of A Chunks:
[1.78070995 2.05785408]

Mean of B Chunks:
[1.42052628 1.99563472 2.34168504]
```

In [21]:
```
step = mean_i__ - mean_total
SSA = n * b * np.matmul(step, step.T)

print('SSA = ' + str(SSA))
```

```
SSA = 2.073839407871998
```

In [22]:
```
step = mean__j_ - mean_total
SSB = n * a * np.matmul(step, step.T)
print('SSB = ' + str(SSB))

mean_1j_ = np.mean(data_a, axis = 0)
mean_2j_ = np.mean(data_b, axis = 0)
box_means = np.ndarray.tolist(mean_1j_) + np.ndarray.tolist(mean_2j_)
```

```
SSB = 15.588407965253385
```

```
In [23]: SSAB = 0
         for i in range(a):
           for j in range(b):
             unit = box_means[i][j] - mean_i__[i] - mean__j_[j] + mean_total
             SSAB += unit * unit

         SSAB = n * SSAB
         print('SSAB = ' + str(SSAB))
```

```
SSAB = 0.8102694671297749
```

```
In [24]: SSE = 0
         for i in range(a):
           for j in range(b):
             for k in range(n):
               unit = all_data[i][k, j] - box_means[i][j]
               SSE += unit * unit

         print('SSE = ' + str(SSE))

         SST = 0
         for i in range(a):
           for j in range(b):
             for k in range(n):
               unit = all_data[i][k, j] - mean_total
               SST += unit * unit

         print('SST = ' + str(SST))

         df = [a - 1, b - 1, (a - 1) * (b - 1), a * b * (n - 1), a * b * n - 1]

         MSA = SSA / df[0]
         MSB = SSB / df[1]
         MSAB = SSAB / df[2]
         MSE = SSE / df[3]

         F_A = MSA / MSE
         F_B = MSB / MSE
         F_AB = MSAB / MSE
```

```
SSE = 35.295860180409285
SST = 53.76837702066446
```

```
In [25]: print('SSA + SSB + SSAB + SSE = ' + str(SSA + SSB + SSAB + SSE))
         print('SST = ' + str(SST))
```

```
SSA + SSB + SSAB + SSE = 53.768377020664445
SST = 53.76837702066446
```

$$\therefore SSA + SSB + SSAB + SSE = SST$$

In [26]:
```
alpha = 0.01
p_comp_A = 1 - f.cdf(F_A, df[0], df[3])
p_comp_B = 1 - f.cdf(F_B, df[1], df[3])
p_comp_AB = 1 - f.cdf(F_AB, df[2], df[3])
```

In [27]:
```
python_print = False

print('ANOVA Table:')
print()

if python_print:
    print('Source\tSS\tDF\tMS\tF\tp-value')

    print('Iron\t' + str(np.round(SSA, 4)) + '\t' + str(df[0]) + '\t' +
          str(np.round(MSA, 4)) + '\t' + str(np.round(F_A, 4)) +
          '\t' + str(np.round(p_comp_A, 4)))

    print('Mol\t' + str(np.round(SSB, 4)) + '\t' + str(df[1]) + '\t' +
          str(np.round(MSB, 4)) + '\t' + str(np.round(F_B, 4)) +
          '\t' + str(np.round(p_comp_B, 4)))

    print('Int.\t' + str(np.round(SSAB, 4)) + '\t' + str(df[2]) +
          '\t' + str(np.round(MSAB, 4)) + '\t' + str(np.round(F_AB)) +
          '\t' + str(np.round(p_comp_AB, 4)))

    print('Error\t' + str(np.round(SSE, 4)) + '\t' + str(df[3]) +
          '\t' + str(np.round(MSE, 4)))

    print('Total\t' + str(np.round(SST, 4)) + '\t' + str(df[4]))

else:
    # For pretty-printing the above table
    import pandas as pd

    data = {'SS': ['2.0738', '15.5884', '0.8103', '35.2959', '53.7684'],
            'DF': ['1', '2', '2', '102', '107'],
            'MS': ['2.0738', '7.7942', '0.4051', '0.346', '-'],
            'F': ['5.9931', '22.5241', '1.0', '-', '-'],
            'p-value': ['0.0161', '0.0', '0.3143', '-', '-']
            }

    dataframe = pd.DataFrame(data,
```

```
                        columns = ['SS', 'DF', 'MS', 'F', 'p-value'],
                        index = ['Iron', 'Molarity', 'Interact', 'Error', 'Total'])
        print(dataframe)
```

```
ANOVA Table:

                SS   DF       MS        F p-value
Iron        2.0738    1   2.0738   5.9931  0.0161
Molarity   15.5884    2   7.7942  22.5241     0.0
Interact    0.8103    2   0.4051      1.0  0.3143
Error      35.2959  102    0.346        -       -
Total      53.7684  107        -        -       -
```

0.4 *Problem* 4

In [28]:
```
A1_1 = [7.80, 7.10, 7.89, 7.82, 9.00, 8.43, 7.65, 7.70,
        7.28, 8.96, 7.75, 7.80, 7.60, 7.00, 7.82, 7.80]
A1_2 = [90.4, 88.9, 85.9, 88.8, 82.50, 92.40, 82.40, 87.40,
        79.60, 95.10, 90.20, 88.00, 94.1, 86.6, 85.9, 88.8]

A2_1 = [7.12, 7.06, 7.45, 7.45, 8.19, 8.25, 7.45, 7.45,
        7.15, 7.15, 7.70, 7.45, 7.06, 7.04, 7.52, 7.70]
A2_2 = [85.1, 89.0, 75.9, 77.9, 66.0, 74.5, 83.1, 86.4,
        81.2, 72.0, 79.9, 71.9, 81.2, 79.9, 86.4, 76.4]

A1_1 = np.asarray(A1_1)
A1_2 = np.asarray(A1_2)
A2_1 = np.asarray(A2_1)
A2_2 = np.asarray(A2_2)
```

In [29]:
```
A1 = np.asmatrix([A1_1, A1_2]).T
A2 = np.asmatrix([A2_1, A2_2]).T
```

In [30]:
```
p = 2
a = 2
b = 4
n = 4

all_data = [A1, A2]
mean_total = np.ravel(np.mean(np.vstack(all_data), axis = 0))

print('Mean of all observations: ' + str(mean_total))
print()

mean_1__ = np.mean(A1, axis = 0)
mean_2__ = np.mean(A2, axis = 0)

mean_i__ = np.vstack([mean_1__, mean_2__])
```

```
        print('Mean of A Chunks: ')
        print(mean_i__)
        print()

        mean__j_ = []

        print('Mean of B Chunks:')
        print()
        for index in range(b):
          blob = np.vstack([A1[4 * index : 4 * index + 4], A2[4 * index : 4 * index + 4]])
          mean__j_.append(np.ravel(np.mean(blob, axis = 0)))
          print('mean[j = ' + str(index + 1) + '] = ' + str(mean__j_[-1]))
```

```
Mean of all observations: [ 7.6434375 83.55625  ]

Mean of A Chunks:
[[ 7.8375   87.9375  ]
 [ 7.449375 79.175   ]]

Mean of B Chunks:

mean[j = 1] = [ 7.46125 85.2375 ]
mean[j = 2] = [ 8.015   81.8375]
mean[j = 3] = [ 7.655   82.2375]
mean[j = 4] = [ 7.4425 84.9125]
```

```
In [31]: mean_ij_ = []
         print('Means of i, j chunks:')
         print()

         for i in range(a):
           mean_ij_.append([])
           for j in range(b):
             blob = all_data[i][4 * j : 4 * j + 4]
             mean_ij_[i].append(np.ravel(np.mean(blob, axis = 0)))
             print('mean[i = ' + str(i + 1) + '][j = ' + str(j + 1) + '] = '
                   + str(mean_ij_[i][j]))
```

```
Means of i, j chunks:

mean[i = 1][j = 1] = [ 7.6525 88.5   ]
mean[i = 1][j = 2] = [ 8.195 86.175]
mean[i = 1][j = 3] = [ 7.9475 88.225 ]
mean[i = 1][j = 4] = [ 7.555 88.85 ]
mean[i = 2][j = 1] = [ 7.27   81.975]
mean[i = 2][j = 2] = [ 7.835 77.5   ]
mean[i = 2][j = 3] = [ 7.3625 76.25   ]
```

```
mean[i = 2][j = 4] = [ 7.33  80.975]
```

```
In [32]: H_A = np.zeros([p, p])

         for i in range(a):
           step = mean_i__[i, :] - mean_total
           H_A += np.matmul(step.T, step)
         H_A = n * b * H_A

         print('H_A = ')
         print()
         print(H_A)
```

```
H_A =

[[  1.20512813  27.2075625 ]
 [ 27.2075625  614.25125   ]]
```

```
In [33]: H_B = np.zeros([p, p])

         for j in range(b):
           unit = np.asmatrix(mean__j_[j] - mean_total)
           H_B += np.matmul(unit.T, unit)

         H_B = n * a * H_B
         print('H_B = ')
         print()
         print(H_B)
```

```
H_B =

[[ 1.69408438 -9.8615625 ]
 [-9.8615625  74.87375   ]]
```

```
In [34]: H_AB = np.zeros([p, p])

         for i in range(a):
           for j in range(b):
             unit = mean_ij_[i][j] - mean_i__[i] - mean__j_[j] + mean_total
             unit = np.asmatrix(unit)
             H_AB += np.matmul(unit.T, unit)

         H_AB = n * H_AB
         print('H_AB = ')
         print()
         print(H_AB)
```

```
H_AB =

[[ 0.13238437  1.5845625 ]
 [ 1.5845625  32.24375   ]]
```

```
In [35]: E = np.zeros([p, p])

         for i in range(a):
           for j in range(b):
             for k in range(n):
               unit = np.asmatrix(all_data[i][4 * j + k] - mean_ij_[i][j])
               E += np.matmul(unit.T, unit)

         print('E = ')
         print()
         print(E)
```

```
E =

[[  4.896525  -1.88975 ]
 [ -1.88975  736.39    ]]
```

```
In [36]: T = np.zeros([p, p])

         for i in range(a):
           for j in range(b):
             for k in range(n):
               unit = np.asmatrix(all_data[i][4 * j + k] - mean_total)
               T += np.matmul(unit.T, unit)

         print('T = ')
         print()
         print(T)
```

```
T =

[[   7.92812187   17.0408125 ]
 [  17.0408125  1457.75875   ]]
```

```
In [37]: # Tests
         nu_H = a - 1
         nu_E = a * b * (n - 1)
         multiplier = (nu_E + nu_H - p) / p
         alpha = 0.05

         # Test for interaction
```

```python
        cap_AB = np.linalg.det(E) / np.linalg.det(E + H_AB)
        F_AB = ((1 - cap_AB) / cap_AB) * multiplier
        F_comp_AB = f.isf(alpha, p, nu_E + nu_H - p)

        print('cap_AB = ' + str(cap_AB))
        print('Calculated F_AB = ' + str(F_AB))
        print('F_AB for comparison = ' + str(F_comp_AB))
        print('Calculated F_AB > F_AB for comparison: ' + str(F_AB > F_comp_AB))

        # Test for A's effect
        cap_A = np.linalg.det(E) / np.linalg.det(E + H_A)
        F_A = ((1 - cap_A) / cap_A) * multiplier
        F_comp_A = f.isf(alpha, p, nu_E + nu_H - p)

        print()
        print('cap_A = ' + str(cap_A))
        print('Calculated F_A = ' + str(F_A))
        print('F_A for comparison = ' + str(F_comp_A))
        print('Calculated F_A > F_A for comparison: ' + str(F_A > F_comp_A))

        # Test for B's effect
        cap_B = np.linalg.det(E) / np.linalg.det(E + H_B)
        F_B = ((1 - cap_B) / cap_B) * multiplier
        F_comp_B = f.isf(alpha, p, nu_E + nu_H - p)

        print()
        print('cap_B = ' + str(cap_B))
        print('Calculated F_B = ' + str(F_B))
        print('F_B for comparison = ' + str(F_comp_B))
        print('Calculated F_B > F_B for comparison: ' + str(F_B > F_comp_B))
```

```
cap_AB = 0.9319287786432168
Calculated F_AB = 0.839998789116378
F_AB for comparison = 3.422132207861178
Calculated F_AB > F_AB for comparison: False

cap_A = 0.47396164219167936
Calculated F_A = 12.763566871829632
F_A for comparison = 3.422132207861178
Calculated F_A > F_A for comparison: True

cap_B = 0.6915794538595539
Calculated F_B = 5.128602738009366
F_B for comparison = 3.422132207861178
Calculated F_B > F_B for comparison: True
```

Conclusions :

- $No\ Factor\ A\ \ effects.$
- $No\ Factor\ B\ \ effects.$
- $Interaction\ effects\ exist.$

```
In [38]: # ^_^ Thank You
```