이승우 (2022311154)

HW5 Report

2022-04-07

1. (a) Assuming that the left and right side of the table are 'somehow' matched pair samples, we rearrange the 'lumber.dat' data into the following form and save it as 'lumber'. The first two columns and the last two columns are stored in the variables 'lumber1' and 'lumber2', respectively.

```
In [3]: lumber1=lumber_raw.iloc[0:15,:]
        lumber1.columns=['x1','x2']
        lumber2=lumber_raw.iloc[15:,:]
        lumber2.reset_index(inplace = True, drop = True)
        lumber2.columns=['x1','x2']
        lumber=pd.concat([lumber1,lumber2],axis=1)
        lumber
```

Out[3]:

	x1	x2	x1	x2
0	1232	4175	1712	7749
1	1115	6652	1932	6818
2	2205	7612	1820	9307
3	1897	10914	1900	6457
4	1932	10850	2426	10102
5	1612	7627	1558	7414
6	1598	6954	1470	7556
7	1804	8365	1858	7833
8	1752	9469	1587	8309
9	2067	6410	2208	9559
10	2365	10327	1487	6255
11	1646	7320	2206	10723
12	1579	8196	2332	5430
13	1880	9709	2540	12090
14	1773	10370	2322	10072

```
In [9]: my_hotelling(lumber_diff,d0)
        #cannot reject null hypothesis
```

```
Out[9]: {'t2': 2.734908183617629,
         'f-value': 1.2697787995367562,
         'p-value': 0.3135310532015403}
```

Using subtraction technique, we save the difference between 'lumber1' and 'lumber2' into a new variable 'lumber_diff'. After defining functions for calculating the mean vector and the covariance matrix, we use the Hotelling's T^2 test function defined in the code from #1 in HW#4. By taking 'lumber_diff' and $\underset{\sim}{\delta}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ as input we conduct the hypothesis test for $H_0: \underset{\sim}{\delta} = \underset{\sim}{0}$ vs. $H_a: \underset{\sim}{\delta} \neq \underset{\sim}{0}$. The p-value is $0.3135 > 0.05$, so at 0.05 significance level, we cannot reject the null hypothesis.

(b) The simultaneous confidence interval for δ_1 is given by the equation

$$\bar{d}_1 \pm \sqrt{\frac{(n-1)p}{n-p} F_{p,n-p}(\alpha)} \sqrt{\frac{s_{d_1}^2}{n}}$$

This is implemented in the following code:

```
In [10]: #(b)
         def sim_conf_int(df,n,p,alpha):
             d_bar = mean_vector(df).iloc[0,0]
             S2=cov_matrix(df).iloc[0,0]
             term2 = np.sqrt((n-1)*p/(n*(n-p))*f.isf(alpha,p,n-p)*S2)
             interval=[d_bar-term2, d_bar+term2]
             return interval
```

```
In [11]: sim_conf_int(lumber_diff,15,2,0.05)
```

```
Out[11]: [-546.0898558358173, 159.2898558358173]
```

The resulting 95% simultaneous confidence interval is $[-546.089, 159.289]$.

(c) Using the profile analysis function defined in the code from #4 in HW#4, we perform a hypothesis test for

$H_0: C\underset{\sim}{M} = \underset{\sim}{0}$ vs. $H_a: C\underset{\sim}{M} \neq \underset{\sim}{0}$

where $\underset{\sim}{M} = \begin{bmatrix} M_{11} \\ M_{12} \\ M_{21} \\ M_{22} \end{bmatrix}$ and $C = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$

The test statistic is $T^2 = n(C\bar{X})'(CSC')^{-1}(C\bar{X}) \overset{H_0}{\sim} \frac{(n-1)p}{n-p} F_{p,n-p}$

The corresponding hypothesis test is implemented in the code below:

In [12]:
```
#(c)
def prof_analysis(df,C):
    n = len(df.index)
    p = len(C.index)
    x_bar = mean_vector(df)
    S = cov_matrix(df)
    t2 = n*((C.dot(x_bar.T)).T.dot(np.linalg.inv(C.dot(S).dot(C.T))).dot(C.dot(x_bar.T))).iloc[0,0]
    fvalue = t2*(n-p)/((n-1)*p)
    pvalue = f.sf(fvalue, p, n-p)
    return {'t2': t2, 'f-value': fvalue, 'p-value': pvalue}
```

In [14]:
```
C1 = pd.DataFrame([[1,0,-1,0],[0,1,0,-1]],columns=['x1','x2','x1','x2'])
C1
```

Out[14]:

	x1	x2	x1	x2
0	1	0	-1	0
1	0	1	0	-1

In [15]:
```
prof_analysis(lumber,C1)
#same result as (a)
```

Out[15]:
```
{'t2': 2.7349081836176325,
 'f-value': 1.269778799536758,
 'p-value': 0.31353105320153973}
```

The result is the same as #1 (a), so we cannot reject the null hypothesis.

2. Assuming equal covariances, we define the function 'two_sample_hotelling' which performs a hypothesis test for $H_0: \underset{\sim}{M}_1 = \underset{\sim}{M}_2$ vs. $H_1: \underset{\sim}{M}_1 \neq \underset{\sim}{M}_2$. The test statistic is

$$T^2 = \frac{n_1 n_2}{n_1+n_2}(\bar{X}_1 - \bar{X}_2)' S_p^{-1}(\bar{X}_1 - \bar{X}_2) \quad \text{where} \quad \frac{n_1+n_2-p-1}{p(n_1+n_2-2)} T^2 \overset{H_0}{\sim} F_{p,\, n_1+n_2-p-1}$$

The corresponding code is shown below:

In [16]:
```
#2
#Hotelling's T^2 test
def two_sample_hotelling(df):
    n1=int(len(df.index)/2) #number of observations for X1
    n2=n1
    p=len(df.columns)-1 #number of variables
    x1=df.iloc[0:n1,:p]
    x2=df.iloc[n2:len(df.index),:p]
    x2.reset_index(inplace = True, drop = True)
    x1_bar=mean_vector(x1)
    x2_bar=mean_vector(x2)
    S1=cov_matrix(x1)
    S2=cov_matrix(x2)
    Sp=((n1-1)*S1+(n2-1)*S2)/(n1+n2-2) #pooled covariance matrix
    t2 = n1*n2/(n1+n2)*(x1_bar-x2_bar).dot(np.linalg.inv(Sp)).dot((x1_bar-x2_bar).T).iloc[0,0] #test statistic
    fvalue = (n1+n2-p-1)/(p*(n1+n2-2))*t2
    pvalue = f.sf(fvalue, p, n1+n2-p-1)
    return {'t2': t2, 'f-value': fvalue, 'p-value': pvalue}
```

3. (a) After saving 'turtle.dat' data as 'turtle', we use the function defined in the code in #2.

In [18]:
```
two_sample_hotelling(turtle)
#p-value < 0.05
#Thus, we reject the null hypothesis
```

Out[18]:
```
{'t2': 72.38162304698841,
 'f-value': 23.07819865266297,
 'p-value': 3.9667264134848684e-09}
```

In our function 'two_sample_hotelling', the input data is separated into two different dataframes based on turtle gender (male / female). 'Female' data corresponds to X1 and 'male' data corresponds to X2.

p-value = $3.9669e-9 < 0.05$, so we reject the null hypothesis ($H_0: \underset{\sim}{M}_1 = \underset{\sim}{M}_2$). At significance level 0.05, the carapace measurements of the female and male turtles are not equal.

(b) The directions of the axes of the confidence ellipsoid for $\underset{\sim}{\mu}_1 - \underset{\sim}{\mu}_2$ are the eigenvectors of S_p. Their half-lengths can be calculated by $\sqrt{\lambda_i}\sqrt{\left(\frac{1}{n_1}+\frac{1}{n_2}\right)c^2}$ where λ_i's are the eigenvalues of S_p and $c^2 = \frac{p(n_1+n_2-2)}{n_1+n_2-p-1} F_{p,\, n_1+n_2-p-1}(\alpha)$. We define the function 'conf_region' shown below which yields the lengths and the directions of the axes of the $100(1-\alpha)\%$ confidence ellipsoid. Notice that we multiply 2 to the half-lengths to calculate the lengths of the axes.

In [19]:
```
#(b) Determine lengths and directions for the axes of 95% confidence elipsoid
def conf_region(df,alpha):
    n1=int(len(df.index)/2) #number of observations for X1
    n2=n1
    p=len(df.columns)-1 #number of variables
    x1=df.iloc[0:n1,:p]
    x2=df.iloc[n2:len(df.index),:p]
    x2.reset_index(inplace = True, drop = True)
    S1=cov_matrix(x1)
    S2=cov_matrix(x2)
    Sp=((n1-1)*S1+(n2-1)*S2)/(n1+n2-2) #pooled covariance matrix
    eigenval = np.linalg.eig(Sp)[0] #eigenvalues
    eigenvec = np.linalg.eig(Sp)[1] #eigenvectors
    c2 = p*(n1+n2-2)/(n1+n2-p-1)*f.isf(alpha,p,n1+n2-p-1)
    halflengths = np.sqrt(eigenval)*np.sqrt((1/n1+1/n2)*c2)
    lengths = 2*halflengths
    return 'lengths:', lengths, 'directions for the axes:', eigenvec
```

In [20]:
```
conf_region(turtle,0.05)
```

Out[20]:
```
('lengths:',
 array([35.84469186,  3.92249939,  2.68457971]),
 'directions for the axes:',
 array([[-0.82017422, -0.54070785, -0.18694725],
        [-0.49543231,  0.83467139, -0.24056286],
        [-0.28611374,  0.10468375,  0.9524601 ]]))
```

The lengths of the axes of the 95% confidence ellipsoid for $\underset{\sim}{\mu}_1 - \underset{\sim}{\mu}_2$ are 35.844, 3.922, 2.684 and the directions of the axes are $[-0.820, -0.540, -0.186]$, $[-0.495, 0.834, -0.240]$, $[-0.286, 0.104, 0.952]$

(c) The $100(1-\alpha)\%$ simultaneous confidence interval for $\underset{\sim}{a}'(\underset{\sim}{\mu}_1 - \underset{\sim}{\mu}_2)$ for all $\underset{\sim}{a}'$ is

$$\left\{ \underset{\sim}{a}'(\bar{\underset{\sim}{X}}_1 - \bar{\underset{\sim}{X}}_2) \pm c\sqrt{\underset{\sim}{a}'\left(\frac{1}{n_1}+\frac{1}{n_2}\right)S_p\,\underset{\sim}{a}} \right\}$$

Our data has three variables ($p=3$), so we should have three intervals for $\underset{\sim}{a}_1 = \begin{bmatrix}1\\0\\0\end{bmatrix}$, $\underset{\sim}{a}_2 = \begin{bmatrix}0\\1\\0\end{bmatrix}$, $\underset{\sim}{a}_3 = \begin{bmatrix}0\\0\\1\end{bmatrix}$

We implement the code as the following function 'sim_conf_int':

In [21]:
```
#(c) Find 95% simultaneous confidence intervals
def sim_conf_int(df,alpha):
    n1=int(len(df.index)/2) #number of observations for X1
    n2=n1
    p=len(df.columns)-1 #number of variables
    x1=df.iloc[0:n1,:p]
    x2=df.iloc[n2:len(df.index),:p]
    x2.reset_index(inplace = True, drop = True)
    x1_bar=mean_vector(x1)
    x2_bar=mean_vector(x2)
    diff=x1_bar-x2_bar
    S1=cov_matrix(x1)
    S2=cov_matrix(x2)
    Sp=((n1-1)*S1+(n2-1)*S2)/(n1+n2-2) #pooled covariance matrix
    c2 = p*(n1+n2-2)/(n1+n2-p-1)*f.isf(alpha,p,n1+n2-p-1)

    a1=pd.DataFrame([1,0,0], index=diff.columns, columns=['mean'])
    a2=pd.DataFrame([0,1,0], index=diff.columns, columns=['mean'])
    a3=pd.DataFrame([0,0,1], index=diff.columns, columns=['mean'])
    a=[a1,a2,a3]
    conf_int=[]

    for i in range(p):
        term1=a[i].T.dot(diff.T).iloc[0,0]
        term2=np.sqrt(c2*(1/n1+1/n2)*a[i].T.dot(Sp).dot(a[i])).iloc[0,0]
        interval=[term1-term2, term1+term2]
        conf_int.append(interval)

    return conf_int
```

```
In [22]: sim_conf_int(turtle,0.05)

Out[22]: [[7.92688151649465, 37.40645181683867],
          [5.256946590448267, 23.32638674288505],
          [6.044543977389379, 16.622122689277276]]
```

The resultant 95% confidence intervals for the component mean differences are

$[7.926, 37.406]$, $[5.256, 23.326)$, $[6.044, 16.622]$

(d) Bonferroni's $100(1-\alpha)\%$ simultaneous confidence interval for $(\mu_{1i} - \mu_{2i})$ is

$$(\bar{X}_{1i} - \bar{X}_{2i}) \pm t_{n_1+n_2-2,\,(\alpha/2p)} \sqrt{\left(\frac{1}{n_1} + \frac{1}{n_2}\right) s_{pii}}$$

We use each of the diagonal components of the pooled covariance matrix S_p to get three intervals as we can see in the code below:

```
In [23]: #(d) Find 95% Bonferroni confidence intervals
         def bonferroni(df,alpha):
             n1=int(len(df.index)/2) #number of observations for X1
             n2=n1
             p=len(df.columns)-1 #number of variables
             x1=df.iloc[0:n1,:p]
             x2=df.iloc[n2:len(df.index),:p]
             x2.reset_index(inplace = True, drop = True)
             x1_bar=mean_vector(x1)
             x2_bar=mean_vector(x2)
             diff=x1_bar-x2_bar
             S1=cov_matrix(x1)
             S2=cov_matrix(x2)
             Sp=((n1-1)*S1+(n2-1)*S2)/(n1+n2-2) #pooled covariance matrix

             conf_int=[]

             for i in range(p):
                 term1=diff.iloc[0,i]
                 term2=t.isf(alpha/(2*p),n1+n2-2)*np.sqrt((1/n1+1/n2)*Sp.iloc[i,i])
                 interval=[term1-term2, term1+term2]
                 conf_int.append(interval)

             return conf_int

In [24]: bonferroni(turtle,0.05)
         #bonferroni intervals are more accurate than regular simultaneous confidence intervals

Out[24]: [[10.344184867410606, 34.98914846592271],
          [6.738627557458531, 21.844705775874782],
          [6.9118977482998565, 15.7547689183668]]
```

The resultant 95% Bonferroni confidence intervals for the component mean differences are

$[10.344, 34.989]$, $[6.738, 21.844]$, $[6.911, 15.754]$.

Comparing with the result from #3(c), we can observe that Bonferroni confidence intervals are more accurate than ordinary simultaneous confidence intervals.

(e) We define a function 'cov_equal' which performs a hypothesis test on the equality of covariance matrices.

$H_0: \Sigma_1 = \Sigma_2$ vs. $H_a: \Sigma_1 \neq \Sigma_2$

When $S_p = \frac{1}{n_1+n_2-2}\left[(n_1-1)S_1 + (n_2-1)S_2\right]$, the test statistic is

$$M = (n_1+n_2-2)\ln|S_p| - (n_1-1)\ln|S_1| - (n_2-1)\ln|S_2| \quad \text{and the scale factor is}$$

$$C^{-1} = 1 - \frac{2p^2+3p-1}{6(p+1)}\left(\frac{n_1+n_2-2}{(n_1-1)(n_2-1)} - \frac{1}{n_1+n_2-2}\right)$$

Then $MC^{-1} \sim \chi^2_\nu$, $\nu = \frac{p(p+1)}{2}$ and we reject H_0 if $MC^{-1} > \chi^2_\nu(\alpha)$

The following code carries out these calculations and yields the value of MC^{-1} and the corresponding p-value.

In [25]:
```
#(e) Test equality of the two population covariance matrices using your own code
def cov_equal(df):
    n1=int(len(df.index)/2) #number of observations for X1
    n2=n1
    p=len(df.columns)-1 #number of variables
    x1=df.iloc[0:n1,:p]
    x2=df.iloc[n2:len(df.index),:p]
    x2.reset_index(inplace = True, drop = True)
    S1=cov_matrix(x1)
    S2=cov_matrix(x2)
    Sp=((n1-1)*S1+(n2-1)*S2)/(n1+n2-2) #pooled covariance matrix

    M=(n1+n2-2)*np.log(np.linalg.det(Sp))-(n1-1)*np.log(np.linalg.det(S1))-(n2-1)*np.log(np.linalg.det(S2)) #test statistic
    C_inv = 1-(2*p**2+3*p-1)/(6*(p+1))*((n1+n2-2)/((n1-1)*(n2-1))-1/(n1+n2-2)) #scale factor
    MC=M*C_inv

    pvalue = chi2.sf(MC, p*(p+1)/2)

    return {'MC^(-1)': MC, 'p-value': pvalue}
```

In [26]:
```
cov_equal(turtle)
#At 0.05 significance level, p-value=0.000671<0.05
#Thus, we reject the null hypothesis
```

Out[26]:
```
{'MC^(-1)': 23.404913718644007, 'p-value': 0.000671608688820075}
```

p-value = 0.000671 < 0.05. Thus, we reject the null hypothesis. At 0.05 significance level, the population covariance matrices are not equal.

(f) We test the equality of the population covariance matrices using the 'test_cov_oneway' from the 'statsmodels' Python package.

In [27]:
```
#(f) Test equality of the two population covariance matrices using Python package
x1=turtle.iloc[0:24,:3]
x2=turtle.iloc[24:48,:3]
x2.reset_index(inplace = True, drop = True)
S1=cov_matrix(x1)
S2=cov_matrix(x2)

mv.test_cov_oneway([S1,S2],[24,24])
#same result
```

Out[27]:
```
<class 'statsmodels.stats.base.HolderTuple'>
statistic = 3.8991762550673945
pvalue = 0.0006786237151972158
statistic_base = 25.18423464462279
statistic_chi2 = 23.404913718644007
pvalue_chi2 = 0.000671608688820075
df_chi2 = 6.0
distr_chi2 = 'chi2'
statistic_f = 3.8991762550673945
pvalue_f = 0.0006786237151972158
df_f = (6.0, 15331.018867924493)
distr_f = 'F'
tuple = (3.8991762550673945, 0.0006786237151972158)
```

We can see that the values 'statistic_chi2' and 'pvalue_chi2' are the same as MC^{-1} and the p-value obtained in #3(e). Thus, we reject the null hypothesis.