이예찬 (2022311154)

HW4 Report

2022-03-31

1. First, we define two function that calculate the sample mean vector and the sample covariance matrix of a dataset.

In [3]:
```
#1
def mean_vector(data):
    ones = pd.DataFrame({'mean': np.ones(len(data))})
    mean = ones.transpose().dot(data)/len(data)
    return mean
```

In [4]:
```
def cov_matrix(data):
    mean = mean_vector(data)
    mean_rep = pd.concat([mean]*len(data))
    mean_rep.columns = data.columns
    mean_rep.reset_index(inplace = True, drop = True)
    cov = (data-mean_rep).transpose().dot(data-mean_rep)/(len(data)-1)
    return cov
```

Next, we define a function, 'my_hotelling', which implements Hotelling's T^2 test. The test statistic is stored in 't2' and is calculated by the formula: $T^2 = n(\bar{X} - \mu_0)' S^{-1} (\bar{X} - \mu_0)$. We then calculate the corresponding F value using $\frac{n-p}{(n-1)p} T^2 \sim F_{p,n-p}$ under H_0. Thus, we reject H_0 when $\frac{(n-p)}{(n-1)p} T^2 > F_{p,n-p}(\alpha)$. Our function returns T^2, the F value, and the p-value.

In [5]:
```
#Hotelling's T^2 test
def my_hotelling(df,mu):
    n=len(df.index) #number of observations
    p=len(df.columns) #number of variables
    mu_reshape=pd.DataFrame(mu.reshape(1,3), index=['mean'], columns=df.columns)
    diff=mean_vector(df)-mu_reshape
    inv_cov = pd.DataFrame(np.linalg.inv(cov_matrix(df)), index=df.columns, columns=df.columns)
    t2 = n*((diff.dot(inv_cov)).dot(diff.T)).iloc[0,0]
    fvalue = (n-p)*t2/((n-1)*p)
    pvalue = f.sf(fvalue, p, n-p)
    return {'t2': t2, 'f-value': fvalue, 'p-value': pvalue}
```

2.(a) After reading 'college.dat' data, we create a vector 'mu0' containing the given mean values for the hypothesis test. We then implement the Python code in #1. The code results in a p-value = $2.828e-23 < 0.05$. Thus, we reject H_0. At $\alpha = 0.05$ level of significance, $\underset{\sim}{\mu} \neq [500, 50, 30]'$.

In [7]:
```
#array of means to test
mu0=np.array([500,50,30])
mu0
```

Out[7]:
```
array([500,  50,  30])
```

In [8]:
```
my_hotelling(college,mu0)
#result: p-value=2.828e-23 < 0.05
#reject null hypothesis
```

Out[8]:
```
{'t2': 223.31017568489185,
 'f-value': 72.70563859508106,
 'p-value': 2.82809706246472e-23}
```

(b) We use the 'hotelling_t2' function from a Python package and we obtain the same result as 2(a). p-value = $2.828e-23 < 0.05$. Thus, we reject H_0. At $\alpha = 0.05$ level of significance, $\underset{\sim}{\mu} \neq [500, 50, 30]'$.

In [9]:
```
#(b) Using Python package
hotelling.stats.hotelling_t2(college,mu0)
#result: p-value=2.828e-23 < 2.713
#reject null hypothesis (same result)
```

Out[9]:
```
(223.31017568489145,
 72.70563859508093,
 2.8280970624648855e-23,
              X1          X2          X3
X1  5808.059342  597.835204  222.029671
X2   597.835204  126.053729   23.388532
X3   222.029671   23.388532   23.111735)
```

(c) We define a function 'conf_region' to determine the lengths and directions of the confidence ellipsoid for μ. We must obtain the eigenvectors and eigenvalues of the sample covariance matrix S, which we achieve using a Python function. The half-lengths and axes of the confidence ellipsoid are calculated by:

half-length: $\sqrt{\lambda_i}\sqrt{\frac{(n-1)p}{n(n-p)}F_{p,n-p}(\alpha)}$ axes: $\pm\sqrt{\lambda_i}\sqrt{\frac{(n-1)p}{n(n-p)}F_{p,n-p}(\alpha)}\,\underset{\sim}{e}_i$

where $\underset{\sim}{e}_1, \underset{\sim}{e}_2, \cdots, \underset{\sim}{e}_p$ are the eigenvectors and $\lambda_1, \lambda_2, \cdots, \lambda_p$ are the eigenvalues of S. Thus, we obtain the half-lengths and directions of the axes of the 95% confidence ellipsoid for μ using the code below:

In [10]:
```
#(c)
def conf_region(df,n,p,alpha):
    S=cov_matrix(df)
    eigenval = np.linalg.eig(S)[0] #eigenvalues
    eigenvec = np.linalg.eig(S)[1] #eigenvectors
    lengths=np.sqrt((n-1)*p/(n*(n-p))*f.isf(alpha,p,n-p))*np.sqrt(eigenval)
    val=np.sqrt((n-1)*p/(n*(n-p))*f.isf(alpha,p,n-p))*np.multiply(eigenvec,np.sqrt(eigenval).reshape(p,1))
    axes=np.vstack((val,-val))
    return 'half-lengths:', lengths, 'directions for the axes:', axes
```

In [11]:
```
conf_region(college,87,3,0.05)
```

Out[11]:
```
('half-lengths:',
 array([23.72999755,  2.47276833,  1.18250011]),
 'directions for the axes:',
 array([[-2.35853725e+01, -2.46154906e+00, -8.85304423e-01],
        [-2.55791539e-01,  2.45938874e+00, -2.36837177e-02],
        [-4.50521377e-02,  6.69323257e-03,  1.18162262e+00],
        [ 2.35853725e+01,  2.46154906e+00,  8.85304423e-01],
        [ 2.55791539e-01, -2.45938874e+00,  2.36837177e-02],
        [ 4.50521377e-02, -6.69323257e-03, -1.18162262e+00]]))
```

The half-lengths and the directions for the axes are shown in the result of cell #11. The lengths of the axes are simply double that of the half-lengths which are approximately $[41.46, 4.94, 2.36]$.

(d) We define a function for obtaining the simultaneous confidence interval, using

$$\underset{\sim}{a}'\bar{\underset{\sim}{X}} \pm \sqrt{\frac{p(n-1)}{n(n-p)}F_{p,n-p}(\alpha)\,\underset{\sim}{a}'S\underset{\sim}{a}}$$

We calculate the first and second term of the formula for the interval and use them to compute the end points of the interval. The parameter 'coeff' is used to input the value for $\underset{\sim}{a}'$. For $\mu_1 - 2\mu_2 + \mu_3$, we use coeff1 = [1, -2, 1]. We obtain the simultaneous confidence interval at $\alpha = 0.05$ as $[422.051, 462.615]$.

In [12]:
```
#(d)
def sim_conf_int(df,n,p,alpha,coeff):
    a = pd.DataFrame(coeff, index=df.columns, columns=['mean'])
    x_bar = mean_vector(df).T
    S=cov_matrix(df)
    term1 = a.T.dot(x_bar).iloc[0,0]
    term2 = np.sqrt((n-1)*p/(n*(n-p))*f.isf(alpha,p,n-p)*(a.T.dot(S).dot(a)).iloc[0,0])
    interval=[term1-term2, term1+term2]
    return interval
```

In [13]:
```
coeff1=[1,-2,1]
sim_conf_int(college,87,3,0.05,coeff1)
```

Out[13]:
```
[422.05125244523714, 462.6154142214296]
```

3. (a) After reading 'stiff.dat' data, we use the function 'sim_conf_int' which we defined in the Python code in #2(d). This time we use coeff2 = [1, 2, -1, -2] for $M_1 + 2M_2 - M_3 - 2M_4$. The obtained simultaneous confidence interval at $\alpha = 0.05$ is [123.103, 769.097].

```
In [15]: coeff2=[1,2,-1,-2]
         sim_conf_int(stiff2,30,4,0.05,coeff2)
```

```
Out[15]: [123.10266537479538, 769.097334625204]
```

(b) We define a new function 'sim_conf_int_large' which is similar to 'sim_conf_int' but we use the formula for large sample assumption. In this case, the simultaneous confidence interval is obtained by

$$\underset{\sim}{a}'\bar{\underset{\sim}{X}} \pm \sqrt{\chi^2_p(\alpha)}\sqrt{\frac{\underset{\sim}{a}' S \underset{\sim}{a}}{n}}$$

The simultaneous confidence interval obtained at $\alpha = 0.05$ is [161.682, 730.518].

```
In [16]: #(b) Repeat under large sample assumption
         def sim_conf_int_large(df,n,p,alpha,coeff):
             a = pd.DataFrame(coeff, index=df.columns, columns=['mean'])
             x_bar = mean_vector(df).T
             S=cov_matrix(df)
             term1 = a.T.dot(x_bar).iloc[0,0]
             term2 = np.sqrt(chi2.isf(alpha,p))*np.sqrt((a.T.dot(S).dot(a)/n).iloc[0,0])
             interval=[term1-term2, term1+term2]
             return interval
```

```
In [17]: sim_conf_int_large(stiff2,30,4,0.05,coeff2)
```

```
Out[17]: [161.68206247064148, 730.5179375293578]
```

4. We define a function 'prof_analysis' to conduct profile analysis. For the hypothesis test $H_0: C\underset{\sim}{M} = 0$ vs $H_a: C\underset{\sim}{M} \neq 0$ where C is $q \times p$ matrix and $\underset{\sim}{M}$ is $p \times 1$ vector we use the generalized form of the test statistic

$$T^2 = n(C\bar{\underset{\sim}{X}})'(CSC')^{-1}(C\bar{\underset{\sim}{X}}) \sim \frac{(n-1)q}{n-q} F_{q,n-q} \quad \text{under } H_0$$

The corresponding F value is calculated as $\frac{n-q}{(n-1)q} T^2$

Our function returns T^2, the F value, and the p-value.

```
In [18]: #4
         def prof_analysis(df,C):
             n = len(df.index)
             q = 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-q)/((n-1)*q)
             pvalue = f.sf(fvalue, q, n-q)
             return {'t2': t2, 'f-value': fvalue, 'p-value': pvalue}
```

5. (a) A test for flat means calls for the hypothesis

$H_0: M_1 = M_2 = M_3 = M_4$ vs. $H_a: M_i \neq M_j$ for some $i \neq j$

where H_0 can be translated to

$H_0: C\underset{\sim}{M} = 0$ where $C = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$

We implement this test using the function 'prof_analysis' and obtain p-value = 1.721e-13 < 0.05. Thus, we reject H_0. At $\alpha = 0.05$ level of significance, the profile is not flat.

In [19]:
```
#5
#(a) Test for flat means
C_flat = pd.DataFrame([[1,-1,0,0],[0,1,-1,0],[0,0,1,-1]])
C_flat
```

Out[19]:

	0	1	2	3
0	1	-1	0	0
1	0	1	-1	0
2	0	0	1	-1

In [20]:
```
prof_analysis(stiff2,C_flat)
#result: p-value=1.721e-13 < 0.05
#reject null hypothesis
```

Out[20]:
```
{'t2': 254.72120597667998,
 'f-value': 79.05140875138345,
 'p-value': 1.7219150170540697e-13}
```

(b) A test for linear trend calls for the hypothesis

$H_0: M_2 - M_1 = M_3 - M_2,\ M_3 - M_2 = M_4 - M_3$ vs. H_a: H_0 is false

where H_0 can be translated to

$H_0: C\underset{\sim}{M} = 0$ where $C = \begin{bmatrix} 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \end{bmatrix}$

We implement this test using the function 'prof_analysis' and obtain p-value = 9.560 e-13 < 0.05. Thus, we reject H_0. At $\alpha = 0.05$ level of significance, the profile does not have a linear trend.

In [21]:
```
#(b) Test for linear trend
C_linear = pd.DataFrame([[1,-2,1,0],[0,1,-2,1]])
C_linear
```

Out[21]:

	0	1	2	3
0	1	-2	1	0
1	0	1	-2	1

In [22]:
```
prof_analysis(stiff2,C_linear)
#result: p-value=9.560e-13 < 0.05
#reject null hypothesis
```

Out[22]:
```
{'t2': 180.3800161429685,
 'f-value': 87.0800077931572,
 'p-value': 9.560459481929842e-13}
```

(c) The Python function 'mv.test_mvmean' requires $C\underset{\sim}{M}$ as its input which we store in the variable 'flat'. We obtain the same p-value as in #5(a). Thus, we reject H_0 and at significance level 0.05, the profile is not flat.

In [24]:
```
mv.test_mvmean(flat)
#result: p-value=1.721e-13 < 0.05
#reject null hypothesis (same result)
```

Out[24]:
```
<class 'statsmodels.stats.base.HolderTuple'>
statistic = 79.05140875138369
pvalue = 1.721915017054007e-13
df = (3, 27)
t2 = 254.72120597668078
distr = 'F'
tuple = (79.05140875138369, 1.721915017054007e-13)
```

We store the $\underset{\sim}{C_M}$ matrix for the test for linear trend in the variable 'linear'. After using 'mv.test_mvmean', we obtain the same p-value as in #5(b). Thus, we reject H_0 and at significance level 0.05, the profile does not have a linear trend.

In [26]:
```
mv.test_mvmean(linear)
#result: p-value=9.560e-13 < 0.05
#reject null hypothesis (same result)
```

Out[26]:
```
<class 'statsmodels.stats.base.HolderTuple'>
statistic = 87.08000779315768
pvalue = 9.560459481929198e-13
df = (2, 28)
t2 = 180.38001614296948
distr = 'F'
tuple = (87.08000779315768, 9.560459481929198e-13)
```