Problem 1)

In [1]:
```
import math
import random
import statistics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from timeit import default_timer
from numpy import unique, ravel
from numpy import matrix, matmul
from numpy import sqrt, dot, array, diagonal, mean, transpose, eye, diag, ones
from numpy import transpose, diag, dot
from numpy.linalg import svd, inv, qr, det, eig
```

In [2]:
```
df = pd.read_csv("skulldataCSV.csv")
df
```

Out[2]:

	MaxBr	BasHt	BasLength	NasHt	Time
0	131	138	89	49	1
1	125	131	92	48	1
2	131	132	99	50	1
3	119	132	96	44	1
4	136	143	100	54	1
...	...	...	...	...	...
85	134	123	95	52	3
86	136	137	101	54	3
87	133	131	96	49	3
88	138	133	100	55	3
89	138	133	91	46	3

90 rows × 5 columns

In [3]:
```
t1 = []
t2 = []
t3 = []
for i in range(len(df)):
    r = df.iloc[i]
    if r[4] == 1:
        t1.append(list(r[:4]))
    elif r[4] == 2:
        t2.append(list(r[:4]))
    elif r[4] == 3:
        t3.append(list(r[:4]))
t1 = random.sample(t1,15)
t2 = random.sample(t2,15)
t3 = random.sample(t3,15)
```

In [4]:
```
X = array(t1+t2+t3)
```

In [5]:
```
p = 4
# number of observations per level
n_lvl = 15
```

In [6]:
```
def center(__X): # nxp
    __xbar = array([np.mean( __X[:, j] ) for j in range(p)])
    return __X - __xbar
```

In [7]:
```
X_cen = center(X)
```

In [8]:
```
t1_cen, t2_cen, t3_cen = center(X[:15]), center(X[15:30]), center(X[30:])
```

Centered data for first level (time=1):

In [9]:
```
t1_cen
```

Out[9]:
```
array([[ -5.73333333,  -1.93333333,  -5.53333333,  -1.8       ],
       [ -2.73333333,  -0.93333333,  -4.53333333,   3.2       ],
       [  3.26666667, -11.93333333,  -2.53333333,   3.2       ],
       [  4.26666667,   4.06666667,   5.46666667,   0.2       ],
       [  0.26666667,   5.06666667,  -8.53333333,  -0.8       ],
       [ -2.73333333,   1.06666667,   5.46666667,   0.2       ],
       [ -4.73333333,   0.06666667,   4.46666667,   1.2       ],
       [  8.26666667,   3.06666667,  -1.53333333,   0.2       ],
       [ 10.26666667,   7.06666667,   2.46666667,   1.2       ],
       [  3.26666667,  -8.93333333,  -4.53333333,   3.2       ],
       [ -6.73333333,   5.06666667,   3.46666667,  -3.8       ],
       [  1.26666667,   0.06666667,  -4.53333333,   3.2       ],
       [ -0.73333333,  -2.93333333,   6.46666667,  -0.8       ],
       [-11.73333333,  -0.93333333,  -1.53333333,  -5.8       ],
       [  4.26666667,   2.06666667,   5.46666667,  -2.8       ]])
```

1.a) Manova Table

In [10]:
```
W = t1_cen.T.dot(t1_cen) + t2_cen.T.dot(t2_cen) + t3_cen.T.dot(t3_cen)
```

In [11]:
```
T = X_cen.T.dot(X_cen)
B = T-W
```

W, sums of squares and cross products for residuals, has df = 2

In [12]:
```
W
```

Out[12]:
```
array([[ 846.26666667,  120.4       ,  -13.73333333,  202.6       ],
       [ 120.4       , 1031.6       ,  122.33333333,  116.86666667],
       [ -13.73333333,  122.33333333,  948.8       ,   50.66666667],
       [ 202.6       ,  116.86666667,   50.66666667,  382.53333333]])
```

B, sums of squares and cross products for treatment, has df = 36

In [13]:
```
B
```

Out[13]:
```
array([[100.84444444,  31.71111111, -46.71111111,  20.28888889],
       [ 31.71111111,  36.31111111, -57.17777778,  12.22222222],
       [-46.71111111, -57.17777778,  90.17777778, -18.82222222],
       [ 20.28888889,  12.22222222, -18.82222222,   5.37777778]])
```

T, total variation, has df = 14*3 = 42

In [14]:
```
T
```

Out[14]:
```
array([[ 947.11111111,  152.11111111,  -60.44444444,  222.88888889],
       [ 152.11111111, 1067.91111111,   65.15555556,  129.08888889],
       [ -60.44444444,   65.15555556, 1038.97777778,   31.84444444],
       [ 222.88888889,  129.08888889,   31.84444444,  387.91111111]])
```

1.a) Wilks Lambda

$\Lambda^* = \frac{|W|}{|W+B|}$

In [15]:
```
WL = det(W) / det(T)
print(WL)
```

```
0.7844150059623962
```

1.a) test-statistic from wilks lambda

According to table 6.3, with p = 4, g = 3,
(and sum of n_l for all levels = 45)

In [16]:
```
test_statistic = ((45-4-2)/4) * ((1-math.sqrt(WL))/(math.sqrt(WL)))
print(test_statistic)
```

```
1.2585892849255116
```

1.a) Critical value for $F_{2p,2(\sum n_l - p - 2)}(0.05) \approx$ 2.02

In [17]:
```
g = 3
print("v1:", 2*4)
print("v2:", 2 * (45 - 4 - 2))
```

```
v1: 8
v2: 78
```

1.a) Result

Since the 0.05 confidence level cutoff value (critical value) is at least 2.02 (between 2.02 and 2.10), and the test statistic (F-statistic) = $1.26 < 2.02$. I fail to reject the null hypothesis $H_0 : \tau_1 = \tau_2 = \tau_3 = 0$ of no treatment effects (teatment=time), at $\alpha = 0.05$.

In [ ]:

1.b)

Althought I have failed to reject H_0, indicating that I have not concluded that there exists one or more variable(s) that change over time, I will still be doing b) just to be sure

I am assuming that since we are allowed to use R, we are allowed to import modules to calculate things for us

In [19]:
```
from scipy.stats import f_oneway
```

In [20]:
```
for i in range(4):
    name = df.columns[i]
    res = f_oneway(array(t1)[:, i], array(t2)[:, i], array(t2)[:, i], array(t3)[:, i])
    print(f"{name}:\nF-statistic: {res[0]}\nP({res[0].round(3)} > F) = {res[1].round(3)}\n")
```

```
MaxBr:
F-statistic: 1.7366543665436651
P(1.737 > F) = 0.17

BasHt:
F-statistic: 0.6961442163244869
P(0.696 > F) = 0.558

BasLength:
F-statistic: 1.9848966613672494
P(1.985 > F) = 0.127

NasHt:
F-statistic: 0.2103302706729576
P(0.21 > F) = 0.889
```

1.b) Analysis:

BasLength and MaxBr are the most likely variables to change over time. If the sample sizes were larger, perhaps we could conclude that they do change over time.

In [22]:
```
file_name = "skull_data_sample.txt"

with open(file_name, "w+") as f:
    f.write(", ".join(list(df.columns)) + "\n")
    for i in range(len(X)):
        f.write(", ".join([str(num) for num in X[i]])+"\n")
print("done")
```

```
done
```

In [ ]:

In [1]:
```
import math
import random
import statistics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from timeit import default_timer
from numpy import unique, ravel
from numpy import matrix, matmul
from numpy import sqrt, dot, array, diagonal, mean, transpose, eye, diag, ones
from numpy import transpose, diag, dot
from numpy.linalg import svd, inv, qr, det
```

In [2]:
```
df = pd.read_csv("wordparity.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	worddiff	wordsame	Arabicdiff	Arabicsame
0	869.0	860.5	691.0	601.0
1	995.0	875.0	678.0	659.0
2	1056.0	930.5	833.0	826.0
3	1126.0	954.0	888.0	728.0
4	1044.0	909.0	865.0	839.0

sampling

In [4]:
```
sample_inds = random.sample([i for i in range(len(df))], 20)
X = array(df)
X = array([X[i] for i in range(len(X)) if i in sample_inds])
```

In [5]:
```
X
```

Out[5]:
```
array([[ 869. ,  860.5,  691. ,  601. ],
       [ 995. ,  875. ,  678. ,  659. ],
       [1126. ,  954. ,  888. ,  728. ],
       [1044. ,  909. ,  865. ,  839. ],
       [ 925. ,  856.5, 1059.5,  797. ],
       [1172.5,  896.5,  926. ,  766. ],
       [1408.5, 1311. ,  854. ,  986. ],
       [1028. ,  887. ,  915. ,  735. ],
       [1011. ,  863. ,  761. ,  657. ],
       [ 726. ,  674. ,  663. ,  583. ],
       [1225. , 1179. , 1037. ,  905.5],
       [ 975.5,  872.5,  814. ,  735. ],
       [ 945. ,  909. ,  867.5,  754. ],
       [ 747. ,  752.5,  777. ,  687.5],
       [ 919. ,  833. ,  752. ,  611. ],
       [ 751. ,  744. ,  683. ,  553. ],
       [ 751. ,  785. ,  789. ,  735. ],
       [ 767. ,  737.5,  724. ,  639. ],
       [1114. , 1046. , 1081. ,  796. ],
       [ 708. ,  669. ,  657. ,  572.5]])
```

2. a)

To compare reaction times for same vs different types of numbers using Hotellings T^2 statistic, I will be obtaining the test statistic from a matrix of pairwise differences, D. This is done under the assumption that for individuals, that pairwise reaction times follow a bivariate normal distribution.

With X_1 be a $20x2$ matrix with the first column,
X_{1j}, is the 'worddiff' vector, and the second column 'Arabic diff'

X_2 be a $20x2$ matrix of 'wordsame' and 'Arabicsame'

In [6]:
```
n = 20
X1 = array([X[:, 0], X[:, 2]]).T
X2 = array([X[:, 1], X[:, 3]]).T
```

In [7]:
```
D = X1 - X2
```

In [8]:
```
Dbar = array([np.mean(D[:, 0]), np.mean(D[:,1])])
Dbar
```

Out[8]:
```
array([ 79.675, 107.125])
```

S_d:

In [9]:
```
S_d = (1/19) * D.T.dot(D)
S_d
```

Out[9]:
```
array([[11937.48684211,  9602.15789474],
       [ 9602.15789474, 19789.27631579]])
```

Hotellings T^2 for pairwise differences:

In [10]:
```
# T^2 = nD' inv(S) D
T2 = n*Dbar.reshape(1,2).dot(inv(S_d)).dot(Dbar.reshape(2,1))
```

T^2:

In [11]:
```
T2
```

Out[11]:
```
array([[13.705746]])
```

F-value:

In [12]:
```
(38/18)*3.55
```

Out[12]:
```
7.4944444444444445
```

63.59 vs (38/18)F2,18

Since T^2 > 7.49, the null hypothesis H_0 can be rejected, $H_0 : \overline{D} = 0$, indicating that the average reaction time to tell if two numbers are both odd or both even is not the same as the average reaction time for when the numbers have one odd and one even.

2.b)

To compare reaction times for Arabic vs word numbers I will be using the exact same method in part a) but the differences this wime will be $D = X_1 - X_2$

With X_1 be a $20x2$ matrix of 'worddiff' 'wordsame',

X_2 be a $20x2$ matrix of 'Arabicdiff' and 'Arabicsame'

In [13]:
```
# adding underscore on variables this time to not rewrite the earlier ones
X_1 = X[:, :2]
X_2 = X[:, 2:]
```

In [14]:
```
D2 = X_1 - X_2
```

Average Difference in reaction time between Word representation and Arabic representation:

In [19]:
```
D_bar = (1/20) * array([np.sum(D2[:, 0]), np.sum(D2[:, 1])])
print(D_bar)
```

```
[136.275 163.725]
```

In [20]:
```
S_d2 = (1/19) * D.T.dot(D)
```

In [21]:
```
T2_b = 20 * D_bar.dot( inv(S_d2) ).dot( D_bar.reshape(2,1) )
```

In [22]:
```
T2_b
```

Out[22]:
```
array([35.96674029])
```

Since T^2 > 7.5, the null hypothesis H_0 can be rejected, $H_0 : \overline{D} = 0$, the average time an individual takes to distinguish between the number test for Arabic numbers vs numbers.

In [23]:
```
file_name = "reaction_times_sample.txt"

with open(file_name, "w+") as f:
    f.write(", ".join(list(df.columns)) + "\n")
    for i in range(len(X)):
        f.write(", ".join([str(num) for num in X[i]])+"\n")
print("done")
```

```
done
```

In [ ]:

In [1]:
```python
import math
import random
import statistics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from timeit import default_timer
from numpy import unique, ravel
from numpy import matrix, matmul
from numpy import sqrt, dot, array, diagonal, mean, transpose, eye, diag, ones
from numpy import transpose, diag, dot
from numpy.linalg import svd, inv, qr, det
from numpy.linalg import eig
```

Problem 3)

In [2]:
```python
df = pd.read_csv("census.csv")
df.head()
```

Out[2]:

	totpop	prof	emp	gov	homeval
0	2.67	5.71	69.02	30.3	1.48
1	2.25	4.37	72.98	43.3	1.44
2	3.12	10.27	64.94	32.0	2.11
3	5.14	7.44	71.29	24.5	1.85
4	5.54	9.25	74.94	31.0	2.23

In [3]:
```python
df_arr = array(df)
rs_inds = random.sample( [i for i in range(len(df))] , 40)
```

In [4]:
```python
data = array([df_arr[ind] for ind in rs_inds])
```

In [5]:
```python
Xbar = array([np.mean(data[:, i]) for i in range(len(data[0]))])
```

In [6]:
```python
Xbar
```

Out[6]:
```
array([ 4.45425,  4.1215 , 71.812  , 27.3025 ,  1.68475])
```

In [7]:
```python
X = data - Xbar
```

In [8]:
```python
S = np.zeros((5,5))
for i in range(len(X)):
    S += X[i].reshape(5,1).dot(X[i].reshape(1,5))
S /= 40
```

3.a) sample variance-covariance matrix

In [9]:
```python
S.round(3)
```

Out[9]:
```
array([[ 2.36800e+00, -1.29900e+00,  2.67600e+00, -4.47500e+00,
        -7.10000e-02],
       [-1.29900e+00,  1.00240e+01, -2.70400e+00,  1.46710e+01,
         1.21300e+00],
       [ 2.67600e+00, -2.70400e+00,  4.56250e+01, -4.48940e+01,
         3.20000e-01],
       [-4.47500e+00,  1.46710e+01, -4.48940e+01,  1.14797e+02,
         1.18600e+00],
       [-7.10000e-02,  1.21300e+00,  3.20000e-01,  1.18600e+00,
         3.10000e-01]])
```

In [10]:
```python
eigens = eig(S)
```

In [11]:
```python
eigVals = eigens[0]
eigVecs = eigens[1]
```

Eigenvalues and eigenvectors:

In [12]:
```python
eigVals
```

Out[12]:
```
array([138.70354113,  24.59057445,   7.63603919,   2.04321343,
         0.15002368])
```

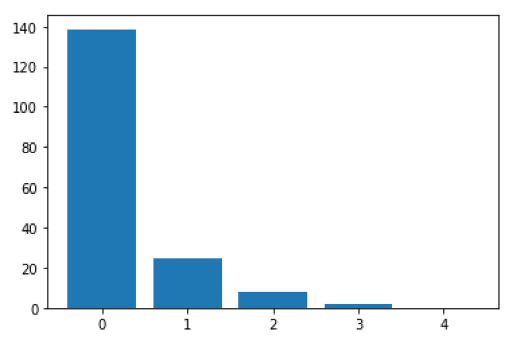
In [13]:
```python
eigVecs
```

Out[13]:
```
array([[-3.88994001e-02,  1.14963032e-02, -1.44197111e-01,
        -9.88319489e-01, -2.80437588e-02],
       [ 1.11380681e-01,  2.41638188e-01,  9.45549446e-01,
        -1.35865040e-01, -1.29159847e-01],
       [-4.34893128e-01,  8.82906969e-01, -1.68633259e-01,
         5.23554262e-02, -1.28446189e-02],
       [ 8.92687383e-01,  4.00110394e-01, -2.07414646e-01,
        -2.23495726e-04,  1.52070034e-04],
       [ 7.64090739e-03,  4.31946450e-02,  1.16984731e-01,
        -4.49904203e-02,  9.91143922e-01]])
```

3.a) screeplot of the eigenvalues

In [14]:
```python
from matplotlib.pyplot import bar
bar([i for i in range(len(eigVals))], eigVals)
```

Out[14]:
```
<BarContainer object of 5 artists>
```



Since 2 eigenvalues explain;

In [15]:
```python
(eigVals[0]+eigVals[1])/np.sum(diag(S))
```

Out[15]:
```
0.9432238694628803
```

I will use 2 the first two eigenvector for the principal components

In [16]:
```python
PCs = eigVecs[:,:2].T.round(4)
for i in range(len(PCs)):
    disp = [str(PCs[i][j])+"X_"+str(j+1) for j in range(5)]
    for k in range(1, len(disp)):
        if disp[k][0] != "-":
            disp[k] = "+"+disp[k]
    disp = " ".join(disp)
    print(f"Y_{i+1} = {disp}")
```

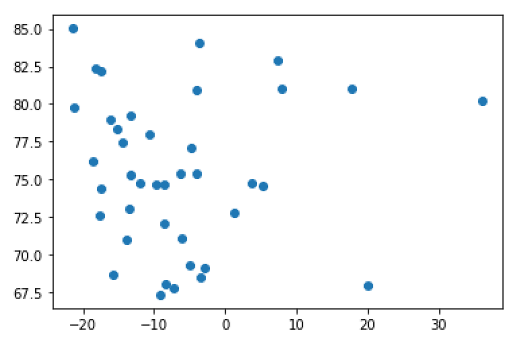
```
Y_1 = -0.0389X_1 +0.1114X_2 -0.4349X_3 +0.8927X_4 +0.0076X_5
Y_2 = 0.0115X_1 +0.2416X_2 +0.8829X_3 +0.4001X_4 +0.0432X_5
```

3.b) scatterplot of the first two principal components

for plotting the Y_1, Y_2's for the data using two principal components

In [17]:
```python
plt.scatter( [PCs[0].dot(obs) for obs in data] , [PCs[1].dot(obs) for obs in data] )
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x1fc4ac6b670>
```



Y1 on x-axis and Y2 on y-axis

In [18]:
```python
# redefine becuse I rounded them earlier for printing
PCs = eigVecs[:,:2].T
```

3.c) 95% Confidence region

As $\alpha = 0.05$ chi squared distribution with two degrees of freedom is 5.99,

95% ellipse has center at,

$\lambda_1 = 138.7, \lambda_2 = 24.6$

following form,

$\frac{Y_1^2}{138.7} + \frac{Y_2^2}{24.6} <= 5.99$

In [ ]:

Problem 4.

A, P are the eigen values and eigenvectors respectively, in the form,
$\Sigma = P \Lambda P'$

In [19]:
```python
A, P = eig(S)
```

pick m = 2 to keep, as the Barlett approximation requirments need m < 2.2984378812835757,

In [20]:
```python
0.5*(11-math.sqrt(41.0))
```

Out[20]:
```
2.2984378812835757
```

4.a)

In [21]:
```python
A, P = A[:2], P[:, :2]
```

$\hat{L}$, the factor loadings:

In [22]:
```python
L = P.dot(diag(A**0.5))
```

In [23]:
```python
L
```

Out[23]:
```
array([[-0.45812783,  0.05700888],
       [ 1.31175777,  1.19825681],
       [-5.12184368,  4.37823715],
       [10.51339959,  1.9841028 ],
       [ 0.08998885,  0.21419743]])
```

$\hat{L}\hat{L}^{prime}$:

In [24]:
```python
LLT = L.dot(L.T)
```

In [25]:
```python
LLT.round(2)
```

Out[25]:
```
array([[ 2.1000e-01, -5.3000e-01,  2.6000e+00, -4.7000e+00, -3.0000e-02],
       [-5.3000e-01,  3.1600e+00, -1.4700e+00,  1.6170e+01,  3.7000e-01],
       [ 2.6000e+00, -1.4700e+00,  4.5400e+01, -4.5160e+01,  4.8000e-01],
       [-4.7000e+00,  1.6170e+01, -4.5160e+01,  1.1447e+02,  1.3700e+00],
       [-3.0000e-02,  3.7000e-01,  4.8000e-01,  1.3700e+00,  5.0000e-02]])
```

Ψ:

In [26]:
```python
psi = diag(S - LLT)
```

In [27]:
```python
psi.round(3)
```

Out[27]:
```
array([2.155, 6.867, 0.223, 0.329, 0.256])
```

4.a)

The Five Communalities

In [28]:
```python
diag(LLT).round(3)
```

Out[28]:
```
array([2.13000e-01, 3.15700e+00, 4.54020e+01, 1.14468e+02, 5.40000e-02])
```

The specific variances

ψ_{ii}

In [29]:
```python
psi
```

Out[29]:
```
array([2.15465331, 6.86732491, 0.22277277, 0.32850889, 0.25601641])
```

b)

How the factor loadings were calculated:

The loadings were calculated using the principal components method, wheren a the eigenvalues and eigenvectors were provided by a spectral decomposition of the variance-covariance matrix of the centered (but not standardized) data.

The spectral decomposition was done using using Python's eig() function, which returns a numpy array of the p eigenvectors and an matrix (array) of the eigenvectors where the columns of the matrix corresponds to the eigenvector.

Factor score for the first observation

(using the scoring metric for when using principal component)

In [30]:
```python
b_left = inv(L.T.dot(L))
b_left.dot(L.T.dot(X[0].reshape(5,1)))
```

Out[30]:
```
array([[-0.61845624],
       [-0.88918844]])
```

sum of factor scores,

Expected to be should be around zero, and is below $9 \cdot 10^{-10}$

In [31]:
```python
f_scores = np.zeros((2,1))
for i in range(40):
    b_left = inv(L.T.dot(L))
    f_scores += b_left.dot(L.T.dot(X[i].reshape(5,1)))
```

In [32]:
```python
f_scores
```

Out[32]:
```
array([[2.63677968e-15],
       [2.13162821e-14]])
```

In [ ]:

In [ ]:

In [34]:
```python
file_name = "census_sample.txt"

with open(file_name, "w+") as f:
    f.write(", ".join(list(df.columns)) + "\n")
    for i in range(len(data)):
        f.write(", ".join([str(num) for num in data[i]])+"\n")
print("done")
```

```
done
```

In [ ]:

In [ ]:

In [1]:
```
import math
import random
import statistics
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from timeit import default_timer
from numpy import unique, ravel
from numpy import matrix, matmul
from numpy import sqrt, dot, array, diagonal, mean, transpose, eye, diag, ones
from numpy import transpose, diag, dot
from numpy.linalg import svd, inv, qr, det
from sklearn.linear_model import LinearRegression
```

In [2]:
```
df = pd.read_csv("bankruptcy.csv")
df.head()
```

Out[2]:

	cftd	nita	cacl	cans	pop
0	-0.45	-0.41	1.09	0.45	0
1	-0.56	-0.31	1.51	0.16	0
2	0.06	0.02	1.01	0.40	0
3	-0.07	-0.09	1.45	0.26	0
4	-0.10	-0.09	1.56	0.67	0

Sampling

In [3]:
```
pop1_inds = [i for i in range(len(df)) if list(df["pop"])[i] == 0]
pop2_inds = [i for i in range(len(df)) if i not in pop1_inds]
pop1_inds = random.sample(pop1_inds, 11)
pop2_inds = random.sample(pop2_inds, 15)
```

In [4]:
```
pop1 = df.iloc[pop1_inds]
pop2 = df.iloc[pop2_inds]
sample = array(df.iloc[pop1_inds + pop2_inds])
```

In [5]:
```
sample[:10]
```

Out[5]:
```
array([[-0.56, -0.31,  1.51,  0.16,  0.  ],
       [ 0.01,  0.  ,  1.26,  0.6 ,  0.  ],
       [ 0.07,  0.02,  1.31,  0.25,  0.  ],
       [-0.28, -0.27,  1.27,  0.51,  0.  ],
       [ 0.05,  0.03,  1.68,  0.95,  0.  ],
       [ 0.15,  0.05,  1.88,  0.27,  0.  ],
       [-0.07, -0.06,  1.37,  0.4 ,  0.  ],
       [ 0.04,  0.01,  1.5 ,  0.71,  0.  ],
       [-0.14, -0.07,  0.71,  0.28,  0.  ],
       [-0.1 , -0.09,  1.56,  0.67,  0.  ]])
```

5.a) Obtaining the linear discriminant function (with intercept)

I will subtract 0.5 from y, 'pop', so that the binary population labels (0, 1) are instead -0.5 and 0.5. (zero centered). The

In [6]:
```
X, y = sample[:, :4], sample[:, 4]
```

In [7]:
```
y_centered = y - 0.5

model = LinearRegression().fit(X, y_centered)
```

In [8]:
```
y_centered
```

Out[8]:
```
array([-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5,  0.5,  0.5,  0.5])
```

In [9]:
```
coef = model.coef_
intercept = model.intercept_
```

5.a) Linear Discriminant Function

By centering the binary population parameter 'pop', normalized least-squares regression coefficients (and intercept) provide a linear discriminant function, with the classification rule,

$y_i < 0 : \pi = 1$

$y_i > 0: \pi = 2$

where y_i is,

In [10]:
```python
ss = " + ".join([f"{coef[i].round(3)}x_{i+1}" for i in range(len(coef))])
print(f"y_i = {ss} {intercept.round(3)}")
```

```
y_i = -0.118x_1 + 2.02x_2 + 0.339x_3 + -0.624x_4 -0.346
```

5.b) Confusion tables and Error Rates

In [11]:
```python
def confusion(Y_pr, Y_ts):
    Y_pr = [1 if obs > 0 else 0 for obs in Y_pr]
    TP, FP, TN, FN = 0, 0, 0, 0
    dd = {(1,1):TP, (1,0):FP, (0,1):FN, (0,0):TN}
    for pr, ts in zip(Y_pr, Y_ts):
        dd[(pr,ts)] += 1
    return {"TP": dd[(1,1)], "FP": dd[(1,0)], "FN": dd[(0,1)], "TN": dd[(0,0)]}
```

In [12]:
```python
ts = array(df.iloc[[i for i in range(len(df)) if i not in pop1_inds + pop2_inds]])
```

In [13]:
```python
X_ts = ts[:, :4]
y_ts = ts[:, 4]
```

what the predictions look like; they get converted to binary (in the confusion function);

In [14]:
```python
y
```

Out[14]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

confusion for the data that was not in my sample:

In [15]:
```python
confusion( model.predict(X), y )
```

Out[15]:
```
{'TP': 15, 'FP': 1, 'FN': 0, 'TN': 10}
```

confusion for the data that was not in my sample:

In [16]:
```python
confusion( model.predict(X_ts), y_ts )
```

Out[16]:
```
{'TP': 9, 'FP': 2, 'FN': 1, 'TN': 8}
```

comparison:

the population classification error rate for the subset that I selected was 3.85% (1/26) while for the subset I did not select it was 15% (3/20). A higher error rate in the unselected sample is expected, as the samples statistics do not reflect the true population statistics, and the data parameters do not reflect the underlying things that affect y. For any system in $Ax = b$ where $A \in \mathbb{R}^{nxp}$, if $n > p$, there exists no perfect solution.

In [23]:
```python
file_name = "bankruptcy_sample.txt"

with open(file_name, "w+") as f:
    f.write(", ".join(list(df.columns)) + ",\n")
    for i in range(len(X)):
        f.write(", ".join([str(num) for num in X[i]])+"\n")
print("done")
```

```
done
```

I delete the comma after because I realize the first line isnt supposed to have a comma

In [ ]:

In [ ]:

Processing math: 100%