Problem1

In [76]:
```
data1 = [0,1,2,4,5,5,7,10,10,12,13,17,39]
```

1.a

In [77]:
```
def mean(data):
    mean1 = sum(data)/len(data)
    return round(mean1,3)
```

In [78]:
```
mean(data1)
```

Out[78]: 9.0

In [79]:
```
def variance(data):
    variance1 = 0
    for i in data:
        variance1 = variance1 + (i-mean(data))**2
    variance1 = variance1/len(data)
    return round(variance1,3)
```

In [80]:
```
variance1 = variance(data1)
```

In [81]:
```
std_dev1 = variance1**0.5
```

In [82]:
```
"%.3f" % std_dev1
```

Out[82]: '9.790'

1.b

In [83]:
```
data1b = sorted(data1)
```

In [84]:
```
data1b
```

Out[84]: [0, 1, 2, 4, 5, 5, 7, 10, 10, 12, 13, 17, 39]

the 1st quatile

In [85]:
```
def first_quartile(data):
    if (len(data) + 1) % 4 == 0:
        return data[(len(data) + 1) / 4 - 1]
    else:
        yushu = (len(data) + 1) % 4
        weishu = (len(data) + 1) / 4
        return (data[weishu-1]*(1-float(yushu)/4) + data[weishu]*(float(yushu)/4))
```

In [86]:
```
"%.3f" % first_quartile(data1b)
```

Out[86]: '3.000'

the 3rd quatile

In [87]:
```
def third_quartile(data):
    if (len(data) + 1)*3 % 4 == 0:
        return data[(len(data) + 1)*3 / 4 - 1]
    else:
        yushu = (len(data) + 1)*3 % 4
        weishu = (len(data) + 1)*3 / 4
        return (data[weishu-1]*(1-float(yushu)/4) + data[weishu]*(float(yushu)/4))
```

In [88]:
```
"%.3f" % third_quartile(data1b)
```

Out[88]: '12.500'

In [89]:
```
interquartile = third_quartile(data1b) - first_quartile(data1b)
```

In [90]:
```
"%.3f" % interquartile
```

Out[90]: '9.500'

1.c

In [91]:
```
def mode(data):
    a = []
    a1 = []
    for i in data:
        if i not in a:
            a.append(i)
    b = [0]*len(a)
    for i in range(len(a)):
        for ii in data:
            if a[i] == ii:
                b[i] += 1
    for i in range(len(b)):
        if b[i] == max(b):
            a1.append(a[i])
    return a1
```

In [92]:
```
mode(data1)
```

Out[92]:
```
[5, 10]
```

1.d

In [93]:
```
f = open("./data.freeway.txt", "r")
```

In [94]:
```
all_text = f.read()
```

In [95]:
```
all_text
```

Out[95]:
```
'Timestamp\tSpeed (mph)\tOccupancy (%)\n05/23/2016 00:00\t62.3\t6.7\n05/23/20
16 00:05\t62.7\t5\n05/23/2016 00:10\t62.5\t5.4\n05/23/2016 00:15\t63.5\t4.5\n
05/23/2016 00:20\t64.3\t4.8\n05/23/2016 00:25\t63.9\t4.2\n05/23/2016 00:30\t6
4.1\t4.2\n05/23/2016 00:35\t63\t3.8\n05/23/2016 00:40\t63.1\t3.4\n05/23/2016
 00:45\t63.7\t3.3\n05/23/2016 00:50\t63.5\t3.3\n05/23/2016 00:55\t63.4\t3\n0
5/23/2016 01:00\t62.8\t3.6\n05/23/2016 01:05\t63\t3.1\n05/23/2016 01:10\t62.8
\t3.7\n05/23/2016 01:15\t63.6\t3\n05/23/2016 01:20\t61.9\t3.6\n05/23/2016 01:
25\t62.5\t2.9\n05/23/2016 01:30\t62.7\t3.1\n05/23/2016 01:35\t63.3\t3\n05/23/
2016 01:40\t61.3\t3.6\n05/23/2016 01:45\t62.9\t2.6\n05/23/2016 01:50\t62.8\t
2.9\n05/23/2016 01:55\t61.4\t3\n05/23/2016 02:00\t60.3\t3.6\n05/23/2016 02:05
\t63.3\t2.8\n05/23/2016 02:10\t63.3\t3\n05/23/2016 02:15\t63.8\t2.6\n05/23/20
16 02:20\t62.9\t3.3\n05/23/2016 02:25\t63\t2.6\n05/23/2016 02:30\t63\t2.7\n0
5/23/2016 02:35\t63.6\t2.7\n05/23/2016 02:40\t63.9\t3\n05/23/2016 02:45\t63.2
\t2.8\n05/23/2016 02:50\t63\t2.7\n05/23/2016 02:55\t63.2\t2.6\n05/23/2016 03:
00\t63.1\t3.2\n05/23/2016 03:05\t63.3\t2.6\n05/23/2016 03:10\t63.4\t2.6\n05/2
3/2016 03:15\t63.1\t2.6\n05/23/2016 03:20\t62.9\t3.3\n05/23/2016 03:25\t63.1
\t2.7\n05/23/2016 03:30\t63.1\t3.3\n05/23/2016 03:35\t63.4\t3.3\n05/23/2016 0
3:40\t63.3\t3.4\n05/23/2016 03:45\t63.4\t3.8\n05/23/2016 03:50\t65\t3.4\n05/2
3/2016 03:55\t63.1\t4\n05/23/2016 04:00\t63.5\t3.1\n05/23/2016 04:05\t64.2\t
```

In [96]:
```
c = all_text.split("\n")
```

In [97]:
```
c.pop(0), c.pop(-1)
```

Out[97]:
```
('Timestamp\tSpeed (mph)\tOccupancy (%)', '')
```

In [98]:
```
Timestamp =[0]*len(c)
Speed = [0]*len(c)
Occupancy = [0]*len(c)
for i in range(len(c)):
    Timestamp[i] = c[i].split("\t")[0]
    Speed[i] = float(c[i].split("\t")[1])
    Occupancy[i] = float(c[i].split("\t")[2])
```

In [99]:
```
def median(data):
    if (len(data)+1)%2 ==0:
        return data[(len(data)+1)/2]
    else:
        return ((data[len(data)/2]+data[len(data)/2+1])/2)
```

In [100]:
```
Speed1 = sorted(Speed)
Q1= first_quartile(Speed1)
Q3 = third_quartile(Speed1)
Mean = mean(Speed1)
Median = median(Speed1)
Mode = mode(Speed1)
```

In [101]:
```
"%.3f" % Q1
```

Out[101]:
```
'35.700'
```

In [102]:
```
"%.3f" % Q3
```

Out[102]:
```
'63.100'
```

In [103]:
```
"%.3f" % Median
```

Out[103]:
```
'44.800'
```

In [104]:
```
"%.3f" % Mean
```

Out[104]:
```
'48.092'
```

In [105]:
```
Mode
```

Out[105]:
```
[63.5]
```

1.e

Since the mean is larger than the median, the distribution has a positive skewness.

Probelm 2

2.a

Since J=q/(q+r+s)

In [106]:
```
J = round((42),3)/(42+30+38)
```

In [107]:
```
"%.3f" % J
```

Out[107]:
```
'0.382'
```

2.b

Suppose Jack, Jim and Mary are the 1st, 2nd and 3rd in the following column

let Y and P be 1, N be 0

In [108]:
```
disease = [0]*3
disease[0] = [1,1,1,1,0,0]
disease[1] = [1,0,1,0,0,0]
disease[2] = [1,1,0,1,0,0]
```

In [109]:
```
disease
```

Out[109]:
```
[[1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0]]
```

For asymmetric data, we use the (r+s)/(q+r+s) to evaluate dissimilarity

since r+s = whole-q-t, and q+r+s = whole-t

In [110]:
```
def dissimilarity(data):
    dissimilarity = [[] for i in range(len(data))]

    for i in range(len(data)):
        for ii in range(len(data)):
            if i > ii :
                q = 0
                t = 0
                for iii in range(len(data[i])):
                    if data[i][iii]==1 and data[ii][iii]==1:
                        q += 1
                    if data[i][iii]==0 and data[ii][iii]==0:
                        t += 1
                dissimilarity[i].append(float((len(data[0])-q-t))/(len(data[0])-t)
            if i == ii:
                dissimilarity[i].append(0)

    return dissimilarity
```

In [111]:
```
dissimilarity(disease)
```

Out[111]:
```
[[0], [0.5, 0], [0.25, 0.75, 0]]
```

We can find the ***1st*** and ***3rd*** patient have the lest disimilarity 0.25

So we can determine they have the same disease

2.c

For category data, we take 3 more features in the data matrix

In [112]:
```
matrix = [[1,0,0],
          [0,1,0],
          [0,0,1],
          [0,1,0]]
```

In [113]:
```
bi_dis = dissimilarity(matrix)
```

Suppose excellent is 3, good is 2, fair is 1

For ordinal variables, use Z = (ri-1)/(M-1), which means: excellemt = (3-1)/(3-1) = 1

good = (2-1)/(3-1) = 0.5

fair = (1-1)/(3-1) = 0

In [114]:
```
nu_matrix = [[1],
             [0],
             [1],
             [0.5]]
```

In [115]:
```
nu_dis = [[] for i in range(len(nu_matrix))]
for i in range(len(nu_matrix)):
    for ii in range(len(nu_matrix)):
        if i > ii :
            for iii in range(len(nu_matrix[i])):
                nu_dis[i].append(((nu_matrix[i][iii]-nu_matrix[ii][iii])**2)**0.5)
        if i == ii:
            nu_dis[i].append(0)
```

In [116]:
```
bi_dis,nu_dis
```

Out[116]:
```
([[0], [1.0, 0], [1.0, 1.0, 0], [1.0, 0.0, 1.0, 0]],
 [[0], [1.0, 0], [0.0, 1.0, 0], [0.5, 0.5, 0.5, 0]])
```

suppose the weights of each feature is 0.5

In [117]:
```
mixed_dis = [[] for i in range(len(nu_dis))]
for i in range(len(nu_dis)):
    for ii in range(len(nu_dis[i])):
        mixed_dis[i].append(bi_dis[i][ii]*0.5 + nu_dis[i][ii]*0.5)
```

In [118]:
```
mixed_dis
```

Out[118]:
```
[[0.0], [1.0, 0.0], [0.5, 1.0, 0.0], [0.75, 0.25, 0.75, 0.0]]
```

2.d

In [119]:
```
def eucl_dist(a,b):
    distance = 0
    for i in range(len(a)):
        distance += (a[i]-b[i])**2
    return round(float(distance)**0.5,3)
```

In [120]:
```
def manh_dist(a,b):
    distance = 0
    for i in range(len(a)):
        distance += abs((a[i]-b[i]))
    return round(distance,3)
```

In [121]:
```
def mink_dist(a,b):
    distance = 0
    for i in range(len(a)):
        if distance < abs((a[i]-b[i])):
            distance = abs((a[i]-b[i]))
    return round(distance,3)
```

In [122]:
```
A = [4,4,2]
B = [-3,2,6]
print "the Euclidean distance is "
print eucl_dist(A,B)
print "the Manhattan distance is "
print manh_dist(A,B)
print "the Minkowski distance is "
print mink_dist(A,B)
```

```
the Euclidean distance is
8.307
the Manhattan distance is
13.0
the Minkowski distance is
7.0
```

2.e

In [123]:
```
def cos_sim(a,b):
    vector = 0
    A_len = 0
    B_len = 0
    for i in range(len(a)):
        vector += a[i]*b[i]
        A_len += a[i]**2
        B_len += b[i]**2
    return round(vector/((float(A_len)**0.5)*(float(B_len)**0.5)),3)
```

In [124]:
```
f2 = open("./home.txt", "r")
```

In [125]:
```
text = f2.read()
```

In [126]:
```
text
```

Out[126]:
```
'Geo-ID\tPlace\t% of Homes Built 2000 to 2009\t% of Homes Built 1990 to 1999
\t% of Homes Built 1980 to 1989\t% of Homes Built 1970 to 1979\t% of Homes Bu
ilt 1960 to 1969\t% of Homes Built 1950 to 1959\t% of Homes Built 1940 to 194
9\t% of Homes Built 1939 or earlier\t% of Homes No bed rooms\t% of Homes 1 be
d rooms\t% of Homes 2 bed rooms\t% of Homes  3 bed rooms\t% of Homes 4 bed ro
oms\t% of Homes 5 or more bed rooms\n16000us0600135\tAcalanes Ridge\t3\t9\t10
\t3\t47\t14\t0\t14\t4\t0\t9\t47\t34\t5\n16000us0600156\tAcampo\t81\t0\t0\t0\t
0\t0\t19\t0\t0\t0\t0\t100\t0\t0\n16000us0600212\tActon\t25\t41\t12\t3\t3\t3\t
1\t12\t0\t1\t11\t43\t36\t9\n16000us0600296\tAdelanto\t33\t8\t8\t6\t3\t1\t2\t3
8\t1\t6\t16\t41\t28\t8\n16000us0600310\tAdin\t4\t26\t8\t9\t24\t3\t17\t8\t0\t8
\t26\t57\t9\t0\n16000us0600394\tAgoura Hills\t4\t37\t36\t10\t4\t2\t1\t7\t1\t8
\t16\t25\t36\t15\n16000us0600450\tAgua Dulce\t20\t28\t16\t5\t3\t3\t4\t21\t0\t
4\t11\t35\t39\t11\n16000us0600464\tAguanga\t43\t15\t29\t0\t0\t0\t0\t12\t0\t35
\t7\t46\t12\t0\n16000us0600478\tAhwahnee\t12\t19\t21\t9\t8\t7\t0\t24\t3\t5\t1
5\t51\t25\t2\n16000us0600562\tAlameda\t5\t11\t16\t16\t6\t7\t35\t4\t3\t20\t35
\t28\t12\t3\n05000us06001\tAlameda County\t8\t11\t17\t14\t13\t8\t21\t7\t4\t18
\t29\t30\t15\t4\n16000us0600618\tAlamo\t12\t22\t22\t12\t15\t8\t1\t9\t0\t1\t7
\t26\t45\t20\n16000us0600674\tAlbany\t5\t7\t8\t7\t8\t12\t39\t14\t0\t18\t48\t2
4\t7\t2\n16000us0600786\tAlderpoint\t15\t17\t41\t0\t0\t0\t26\t0\t15\t0\t0\t54
```

In [127]:
```
line = text.split("\n")
```

In [128]:
```
line.pop(0),line.pop(-1)
```

Out[128]:
```
('Geo-ID\tPlace\t% of Homes Built 2000 to 2009\t% of Homes Built 1990 to 1999
\t% of Homes Built 1980 to 1989\t% of Homes Built 1970 to 1979\t% of Homes Buil
t 1960 to 1969\t% of Homes Built 1950 to 1959\t% of Homes Built 1940 to 1949\t%
 of Homes Built 1939 or earlier\t% of Homes No bed rooms\t% of Homes 1 bed room
s\t% of Homes 2 bed rooms\t% of Homes  3 bed rooms\t% of Homes 4 bed rooms\t% o
f Homes 5 or more bed rooms',
 '')
```

In [129]:
```
Geo_ID = [0]*len(line)
Place = [0]*len(line)
area = [[] for i in range(len(line))]

for i in range(len(line)):
    Geo_ID[i] = line[i].split("\t")[0]
    Place[i] =  line[i].split("\t")[1]
    for ii in range(2,len(line[i].split("\t"))):
        area[i].append(int(line[i].split("\t")[ii]))
```

In [130]:
```
Alto = [0,0,12,34,14,21,13,5,0,30,41,23,3,2]
```

In [131]:
```
cos_ans = [[] for i in range(len(area))]
for i in range(len(area)):
    cos_ans[i].append(cos_sim(Alto,area[i]))
    cos_ans[i].append(Geo_ID[i])
    cos_ans[i].append(Place[i])
```

In [132]:
```
import numpy as np
def sort_by_col(data,icol):
    data1 = np.array(data).tolist()
    data1.sort(key=lambda x:x[icol])
    return np.array(data1)
```

In [133]:
```
ans = sort_by_col(cos_ans, 0)
```

In [134]:
```
for i in range(1,6):
    print ans[-i]
```

```
['0.927' '16000us0604870' 'Bell']
['0.924' '16000us0604996' 'Bell Gardens']
['0.923' '16000us0608954' 'Burbank']
['0.919' '16000us0602980' 'Ashland']
['0.915' '16000us0603209' 'August']
```

--

Problem 3

3.a

In [374]:
```
empl = [[1,2,3,4,5],
        [27,51,52,33,45],
        [19000,64000,100000,55000,45000]]
```

zi=(xi−min(x))/(max(x)−min(x))

In [376]:
```
def norm(data):
    data1 = [0]*len(data)
    for i in range(len(data)):
        data1[i] = round(float((data[i]-min(data)))/(max(data)-min(data)),3)
    return data1
```

In [378]:
```
empl[1] = norm(empl[1])
empl[2] = norm(empl[2])
```

In [379]:
```
empl
```

Out[379]:
```
[[1, 2, 3, 4, 5],
 [0.0, 0.96, 1.0, 0.24, 0.72],
 [0.0, 0.556, 1.0, 0.444, 0.321]]
```

3.b

σ = (sum(X-mean)^2/n)^0.5

z = (x−μ)/ σ

In [139]:
```
def std_norm(data,test):
    m = mean(data)
    pstd = variance(data)**0.5
    z = [0]*len(data)
    for i in range(len(data)):
        z[i] = (data[i] - m)/pstd
    test = (test - m)/pstd
    return z, round(test,3)
```

In [140]:
```
std_speed, speed_55 = std_norm(Speed,55)
```

In [141]:
```
mean(Speed),mean(std_speed)
```

Out[141]: (48.092, -0.0)

In [142]:
```
variance(Speed),variance(std_speed)
```

Out[142]: (200.021, 1.0)

In [108]:
```
round(speed55,3)
```

Out[108]: 0.488

Problem 4

4.I.a

In [204]:
```
data = [[2.5,0.5,2.2,1.9,3.1,2.3,2,1.0,1.5,1.1],[2.4,0.7,2.9,2.2,3.0,2.7,1.6,1.1,1
```

In [205]:
```
A = data[0]
B = data[1]
```

In [206]:
```
A_mean = mean(data[0])
B_mean = mean(data[1])
A_std = variance(data[0])**0.5
B_std = variance(data[1])**0.5
```

In [207]:
```
AB = 0
for i in range(len(A)):
    AB += A[i] * B[i]
```

In [209]:
```
coefficient = (AB-len(A)*A_mean*B_mean)/(len(A)*A_std*B_std)
```

In [211]:
```
round(coefficient,3)
```

Out[211]: 0.926

We can find the x-axis's data is highly related to the y-axis's data

4.I.b

If a data has a high variance in signal while low variance in noises, we need consider use the PCA.

Since the correlation is close to 1, which means x-axis's data is highly related to the y-axis's data, if we use the x-y basis, there would be high redundancy.

If we chose to use PCA, it would change the aspect and show the most difference in noise.

In this case, we should use PCA to do the job.

4.I.c

In [213]:
```
covariance = AB/len(A)-A_mean*B_mean
```

In [215]:
```
round(covariance,3)
```

Out[215]: 0.554

Since the Co-variance is higher than 0,

data in X-axis rise with Y-axis.

4.II

In [331]:
```
a = [[1,-1,0,0],[1,0,0,-1]]

m = len(a)
n = len(a[0])
for i in range(m):
    for ii in range(n):
        a[i][ii] = a[i][ii] - mean(a[i])
```

In [344]:
```
def transpose(data):
    data = [[row[col] for row in data] for col in range(len(data[0]))]
    return data
```

In [345]:
```
def dot_mult(A, B):
    ans = [[0] * len(B[0]) for i in range(len(A))]
    for i in range(len(A)):
        for ii in range(len(B[0])):
            for iii in range(len(B)):
                ans[i][ii] += A[i][iii] * B[iii][ii]
    return ans
```

In [346]:
```
def cov(data):
    n = len(data[0])
    m = len(data)
    cov = [[0]*m for i in range(m)]
    for i in range(m):
        for ii in range(m):
            cov[i][ii] = dot_mult(data,transpose(data))[i][ii]/n
    return cov
```

In [335]:
```
cov_a = cov(a)
```

In [336]:
```
cov_a
```

Out[336]:
```
[[0.5, 0.25], [0.25, 0.5]]
```

In [337]:
```
from sympy import *
def eigenvlaues(data):
    dat = data
    x = Symbol('x')
    for i in range(len(dat)):
        dat[i][i] -= x
    a = Matrix(dat)
    print a.det()
    return solve(a.det(),x)
```

In [338]:
```
e_value = eigenvlaues(cov_a)
```

```
x**2 - 1.0*x + 0.1875
```

In [339]:
```
e_value
```
Out[339]: [0.250000000000000, 0.750000000000000]

When e_value is 0.25, it would be cov_a minus e_value in the diagnol

[[0.25,0.25],[0.25,0.25]]

this means the eigenvector is [1,-1]

When e_value is 0.75, it would be cov_a minus e_value in the diagnol [[-0.25,0.25],[0.25,-0.25]]

this means the eigenvector is [1,1]

In [340]:
```
P = [[1,-1],[1,1]]
```
In [341]:
```
Y = dot_mult(P,a)
```
In [342]:
```
Y
```
Out[342]: [[0.0, -1.0, 0.0, 1.0], [2.0, -1.0, 0.0, -1.0]]

In [349]:
```
cov_y = cov(Y)
```
In [350]:
```
cov_y
```
Out[350]: [[0.5, 0.0], [0.0, 1.5]]

Now the Y become a diagonal matrix

since 1.5>0.5, the ***frist principle vector*** is the 2nd row

if we make it norminalization

In [405]:
```
P1 = [round((1**2+1**2)**0.5/2,3),round((1**2+1**2)**0.5/2,3)]
```
In [406]:
```
P1
```
Out[406]: [0.707, 0.707]

4.II.b

The coordinate should be P[1]*a in 1-d space

In [407]:
```
a_1d = [0]*4
for ii in range(4):
    a_1d[ii] = P1[0]*a[0][ii] + P1[0]*a[1][ii]
```

In [408]:
```
a_1d
```

Out[408]: `[1.414, -0.707, 0.0, -0.707]`

4.II.C

we need to use transposed P[1]*a_1d to recover

In [409]:
```
P_tran = [[0.707],[0.707]]
```

In [413]:
```
a_recover = [[0]*4 for i in range(2)]
for i in range(2):
    for ii in range(4):
        a_recover[i][ii] = round(P_tran[i][0]*a_1d[ii],2)
```

In [414]:
```
a_recover
```

Out[414]: `[[1.0, -0.5, 0.0, -0.5], [1.0, -0.5, 0.0, -0.5]]`

In [412]:
```
a
```

Out[412]: `[[1.0, -1.0, 0.0, 0.0], [1.0, 0.0, 0.0, -1.0]]`

After compare, we can find half of the data(2 of 4) can be recoverd.

In [ ]: