CSC487: Data Mining - Homework #4

Cason Konzer

Please use the table below for questions 1 through 3. Notice that Count column is NOT an attribute. It just tells how many times a row occurs in our database and status is our target variable.

department	age	salary	status	count
sales	31_35	46K_50K	senior	30
sales	26_30	26K_30K	junior	40
sales	31_35	31K_35K	junior	40
systems	21_25	46K_50K	junior	20
systems	31_35	66K_70K	senior	5
systems	26_30	46K_50K	junior	3
systems	41_45	66K_70K	senior	3
marketing	36_40	46K_50K	senior	10
marketing	31_35	41K_45K	junior	4
secretary	46_50	36K_40K	senior	4
secretary	26_30	26K_30K	junior	6

```
# load in the data...
data = {
    'department' : ['sales', 'sales', 'sales', 'systems',
                    'systems', 'systems', 'systems', 'marketing',
                    'marketing', 'secretary', 'secretary'],
    'age' : ['31_35', '26_30', '31_35', '21_25',
             '31_35', '26_30', '41_45', '36_40',
             '31_35', '46_50', '26_30'],
    'salary' : ['46K_50K', '26K_30K', '31K_35K', '46K_50K',
                '66K_70K', '46K_50K', '66K_70K', '46K_50K',
                '41K_45K', '36K_40K', '26K_30K'],
    'status' : ['senior', 'junior', 'junior', 'junior',
                'senior', 'junior', 'senior', 'senior',
                'junior', 'senior', 'junior'],
    'count' : [30, 40, 40, 20, 5, 3, 3, 10, 4, 4, 6]
}
df1 = pd.DataFrame(data)

# preview the data...
df1
```

```
##    department    age   salary  status  count
## 0       sales  31_35  46K_50K  senior     30
## 1       sales  26_30  26K_30K  junior     40
## 2       sales  31_35  31K_35K  junior     40
## 3     systems  21_25  46K_50K  junior     20
## 4     systems  31_35  66K_70K  senior      5
## 5     systems  26_30  46K_50K  junior      3
## 6     systems  41_45  66K_70K  senior      3
## 7   marketing  36_40  46K_50K  senior     10
## 8   marketing  31_35  41K_45K  junior      4
## 9   secretary  46_50  36K_40K  senior      4
## 10  secretary  26_30  26K_30K  junior      6
```

1. Given a data tuple having the values "systems", "26_30", and "46K_50K" for the attributes department, age, and salary, respectively, what would a naive Bayesian classification of the status? *(20 points total)*

Given the data we have there is only one status that fits our situation

```
# mask for the given conditions and preview the satisfactory data.
dpt_mask = df1["department"] == "systems"
age_mask = df1["age"] == "26_30"
sal_mask = df1["salary"] == "46K_50K"
df1[dpt_mask & age_mask & sal_mask]
```

```
##   department    age   salary  status  count
## 5    systems  26_30  46K_50K  junior      3
```

Thus the naive Bayesian classification predicts junior status.

Using Bayes' Theorem, $P(junior|dpt = systems, age = 16\_30, sal = 46K\_50K) = \frac{\frac{1}{6} \cdot \frac{6}{11}}{\frac{1}{11}} = 1.$

---

2. Split your diabetes data into two parts for training and testing purposes. Namely, reserve last 10 rows of the diabetes_train.csv for the test set. Then fit a SVM classifier on the bigger portion of this data and test it on these 10 rows you had reserved. ***(20 points)***

```
# read in the data...
# df2 = pd.read_csv(my_data_path.csv)
```

```
# preview the data...
df2
```

```
##       preg    plas    pres    skin   ...   mass    pedi    age             class
## 0        6     148      72      35   ...    33.6   0.627     50  tested_positive
## 1        1      85      66      29   ...    26.6   0.351     31  tested_negative
## 2        8     183      64       0   ...    23.3   0.672     32  tested_positive
## 3        1      89      66      23   ...    28.1   0.167     21  tested_negative
## 4        0     137      40      35   ...    43.1   2.288     33  tested_positive
## ..     ...     ...     ...     ...   ...     ...     ...    ...              ...
## 753      0     181      88      44   ...    43.3   0.222     26  tested_positive
## 754      8     154      78      32   ...    32.4   0.443     45  tested_positive
## 755      1     128      88      39   ...    36.5   1.057     37  tested_positive
## 756      7     137      90      41   ...    32.0   0.391     39  tested_negative
## 757      0     123      72       0   ...    36.3   0.258     52  tested_positive
##
## [758 rows x 9 columns]
```

```
# split the data into training set...
diabetes_train = df2.iloc[:-10]
diabetes_train
```

```
##       preg    plas    pres    skin   ...   mass    pedi    age             class
## 0        6     148      72      35   ...    33.6   0.627     50  tested_positive
## 1        1      85      66      29   ...    26.6   0.351     31  tested_negative
## 2        8     183      64       0   ...    23.3   0.672     32  tested_positive
## 3        1      89      66      23   ...    28.1   0.167     21  tested_negative
## 4        0     137      40      35   ...    43.1   2.288     33  tested_positive
## ..     ...     ...     ...     ...   ...     ...     ...    ...              ...
## 743      9     140      94       0   ...    32.7   0.734     45  tested_positive
## 744     13     153      88      37   ...    40.6   1.174     39  tested_negative
## 745     12     100      84      33   ...    30.0   0.488     46  tested_negative
## 746      1     147      94      41   ...    49.3   0.358     27  tested_positive
## 747      1      81      74      41   ...    46.3   1.096     32  tested_negative
##
## [748 rows x 9 columns]
```

```
# split the data into testing set...
diabetes_test = df2.iloc[-10:]
diabetes_test
```

```
##       preg    plas    pres    skin   ...   mass    pedi    age             class
## 748      3     187      70      22   ...   36.4   0.408     36  tested_positive
## 749      6     162      62       0   ...   24.3   0.178     50  tested_positive
## 750      4     136      70       0   ...   31.2   1.182     22  tested_positive
## 751      1     121      78      39   ...   39.0   0.261     28  tested_negative
## 752      3     108      62      24   ...   26.0   0.223     25  tested_negative
## 753      0     181      88      44   ...   43.3   0.222     26  tested_positive
## 754      8     154      78      32   ...   32.4   0.443     45  tested_positive
## 755      1     128      88      39   ...   36.5   1.057     37  tested_positive
## 756      7     137      90      41   ...   32.0   0.391     39  tested_negative
## 757      0     123      72       0   ...   36.3   0.258     52  tested_positive
##
## [10 rows x 9 columns]
```

```
# split data sets into attributes & labels
diabetes_train_X = diabetes_train.iloc[:,:-1]
diabetes_train_y = diabetes_train.iloc[:,-1]
diabetes_test_X = diabetes_test.iloc[:,:-1]
diabetes_test_y = diabetes_test.iloc[:,-1]
```

```
# train the model on the training set...
svm_clf = svm.SVC(C=1000, gamma=0.00001, kernel='rbf')
svm_clf.fit(diabetes_train_X, diabetes_train_y)
```

```
## SVC(C=1000, gamma=1e-05)
```

```
# use the model to predict on the test set & evaluate...
svm_clf_predictions = svm_clf.predict(diabetes_test_X)
confusion_matrix(diabetes_test_y, svm_clf_predictions)
```

```
## array([[3, 0],
##        [3, 4]], dtype=int64)
```

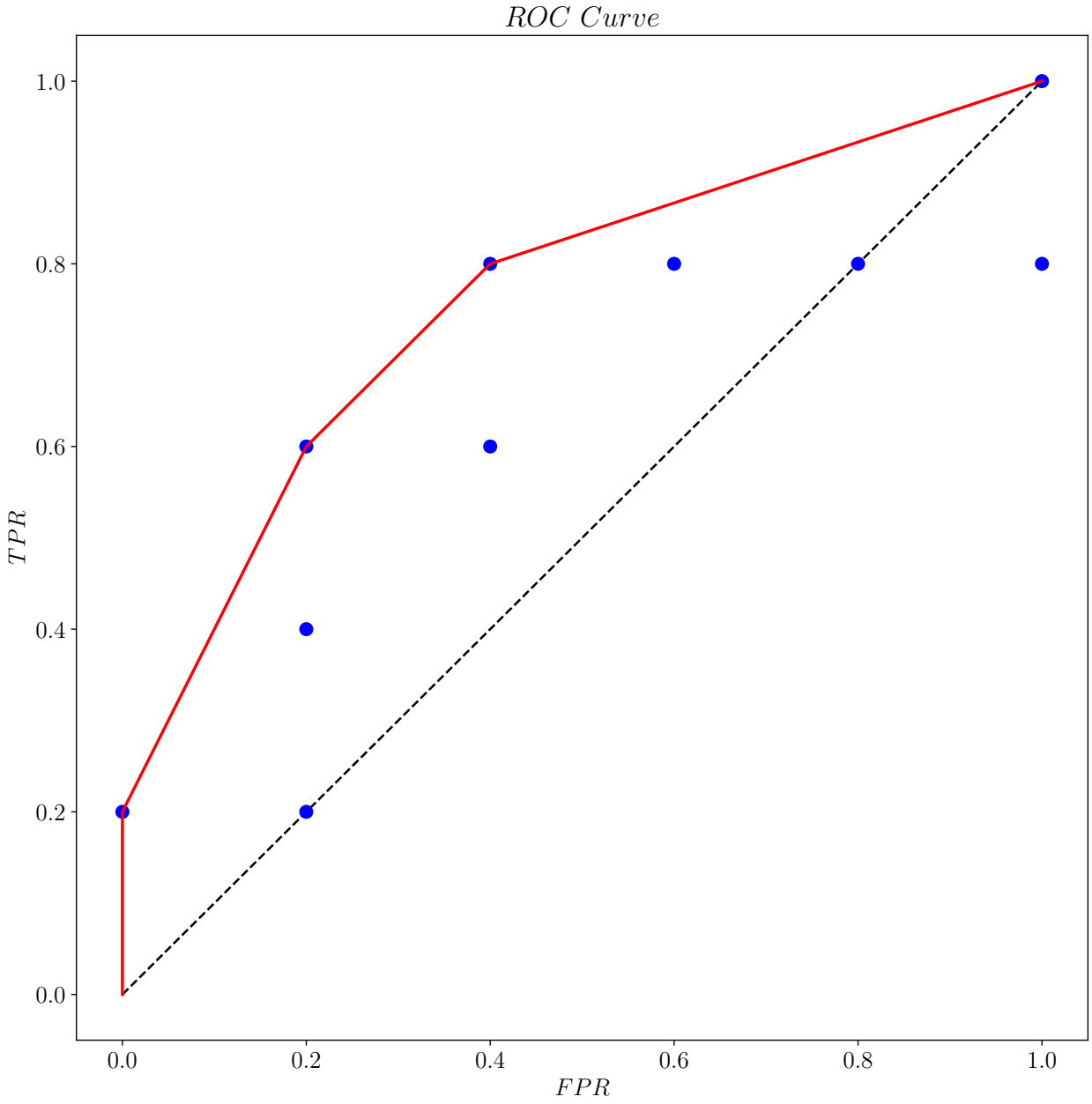
From the confusion matrix we can see 3 true positives, 4 true negatives, and 3 false positives .

---

3. Draw the ROC curve based on the table below and fill the empty columns based on threshold at each step. ***(20 points)***

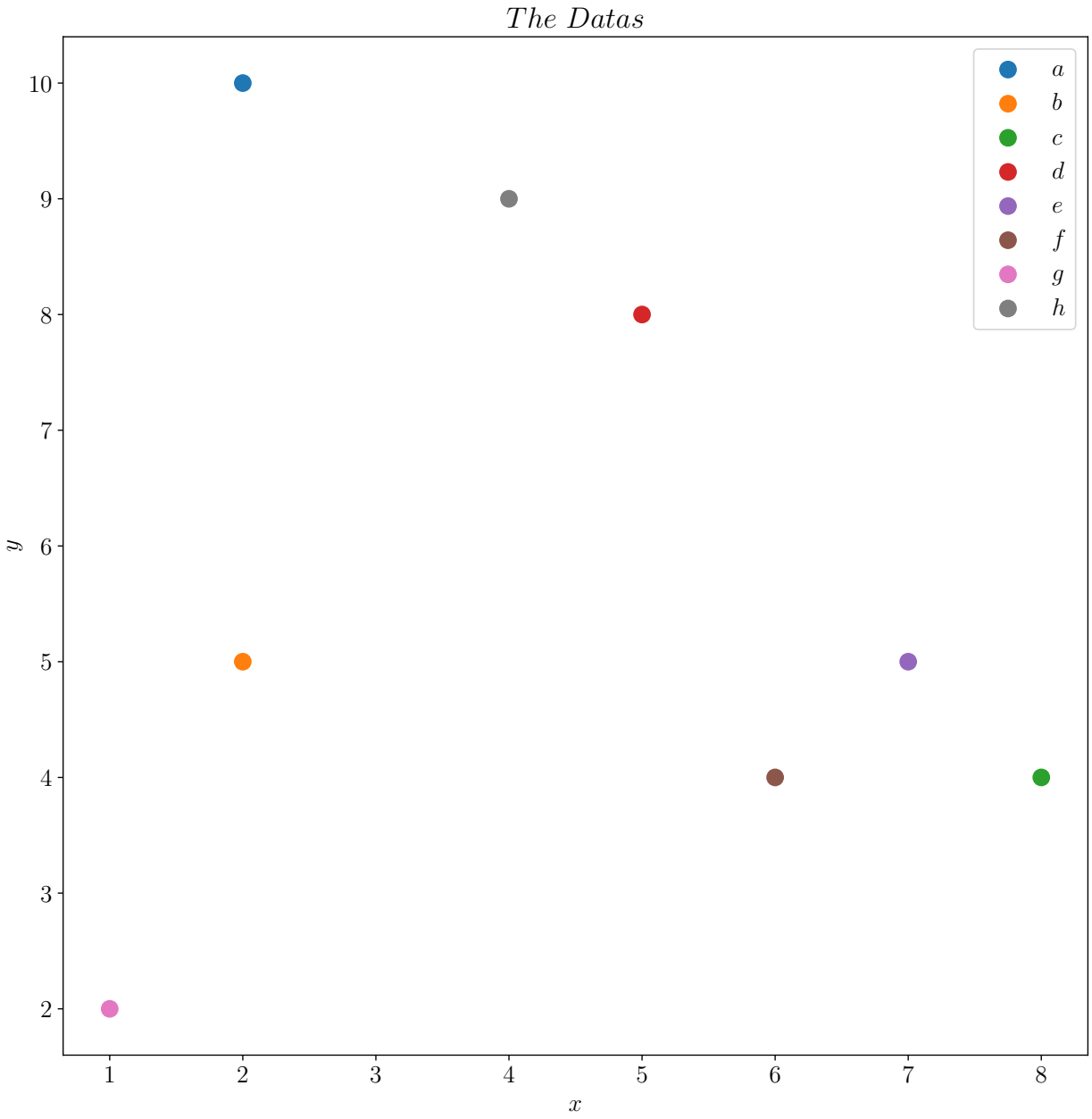
tuple #	class	prob	TP	FN	FP	TN	TPR	FPR
1	p	0.95						
2	n	0.85						
3	p	0.78						
4	p	0.66						
5	n	0.60						
6	p	0.55						
7	n	0.53						
8	n	0.52						
9	n	0.51						
10	p	0.40						

tuple #	class	prob	TP	FN	FP	TN	TPR	FPR
1	p	0.95	1	4	0	5	0.2	0.0
2	n	0.85	1	4	1	4	0.2	0.2
3	p	0.78	2	3	1	4	0.4	0.2
4	p	0.66	3	2	1	4	0.6	0.2
5	n	0.60	3	2	2	3	0.6	0.4
6	p	0.55	4	1	2	3	0.8	0.4
7	n	0.53	4	1	3	2	0.8	0.6
8	n	0.52	4	1	4	1	0.8	0.8
9	n	0.51	4	1	5	0	0.8	1.0
10	p	0.40	5	0	5	0	1.0	1.0


ROC Curve
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

4. Please use the data shown for questions below. ***(20 points)***

	a	b	c	d	e	f	g	h
x	2	2	8	5	7	6	1	4
y	10	5	4	8	5	4	2	9



a.) If h and c are selected as the initial centers for your k-means clustering, assign memberships for other points, and compute the means (centroids) of your initial clusters. You can use Manhattan distance.

```
xs = np.array([2,2,8,5,7,6,1,4])
ys = np.array([10,5,4,8,5,4,2,9])
cx = 8
cy = 4
cx_diff = np.abs(cx -xs)
cy_diff = np.abs(cy -ys)
c_dist = cx_diff + cy_diff
c_dist
```

```
## array([12,  7,  0,  7,  2,  2,  9,  9])
```

```
hx = 4
hy = 9
hx_diff = np.abs(hx -xs)
hy_diff = np.abs(hy -ys)
h_dist = hx_diff + hy_diff
h_dist
```

```
## array([ 3,  6,  9,  2,  7,  7, 10,  0])
```

p_i	a	b	c	d	e	f	g	h
d(p_i, c)	12	7	*	7	2	2	9	*
d(p_i, h)	3	6	*	2	7	7	11	*

We can see that cluster within initial center c contains, {c,e,f,g}, and cluster with initial center h contains, {a,b,d,h}. Moving forward we will reference cluster C_1 as that with initial center c, and cluster C_2 as that with initial center h. To now compute the means, or centroids, of C_1 and C_2, we will take the average x and y values for the points within the clusters. We will reference the centroid of C_1 as z_1 and the center of C_2 as z_2.

$$z_1 = \left(\frac{8+7+6+1}{4}, \frac{4+5+4+2}{4}\right) = (5.5, 3.75).$$

$$z_2 = \left(\frac{2+2+5+4}{4}, \frac{10+5+8+9}{4}\right) = (3.25, 8).$$

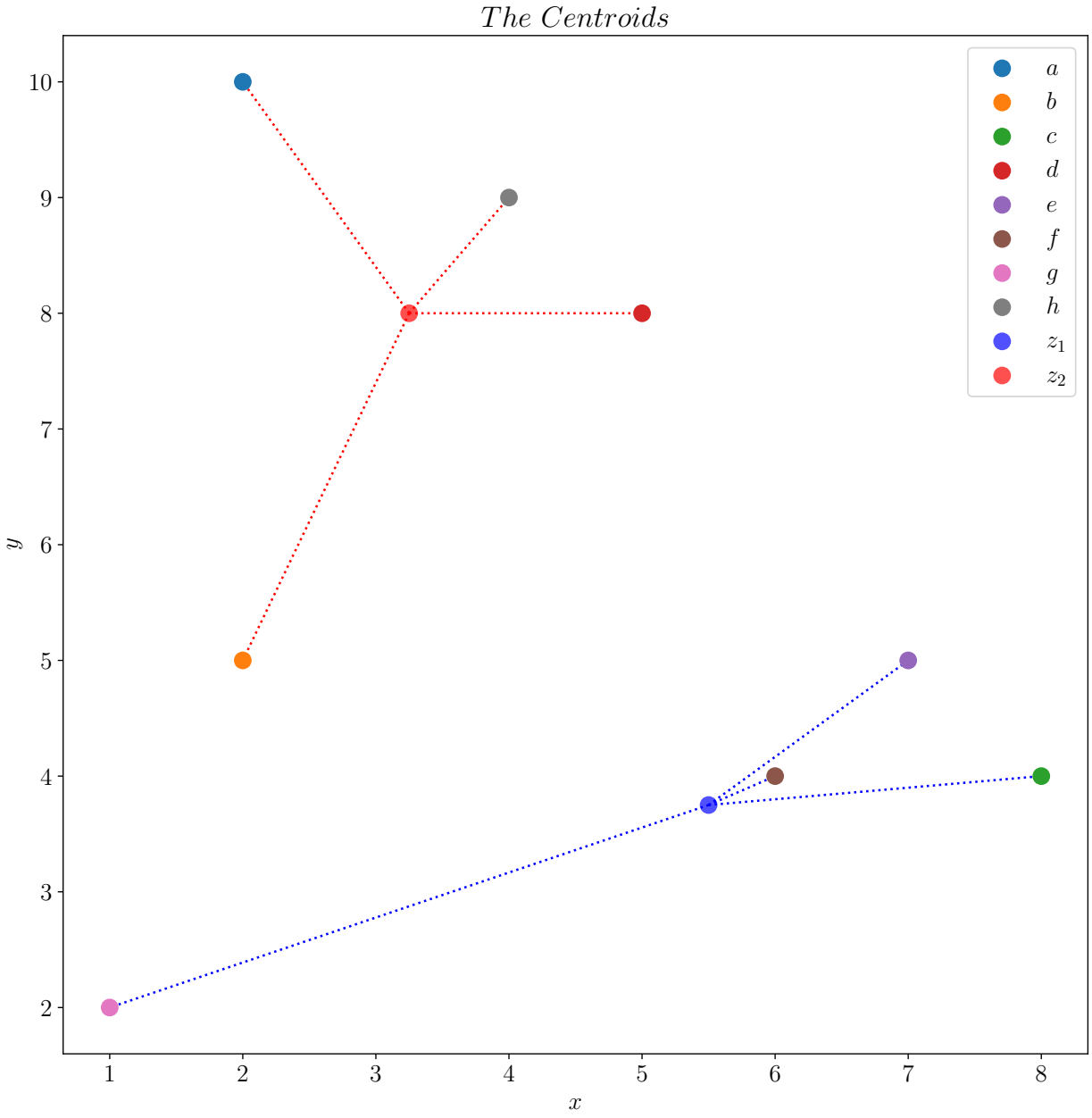
```
z1xs = np.array([8,7,6,1])
z1ys = np.array([4,5,4,2])
z1x = np.mean(z1xs)
z1y = np.mean(z1ys)
z1x, z1y
```

```
## (5.5, 3.75)
```

```
z2xs = np.array([2,2,5,4])
z2ys = np.array([10,5,8,9])
z2x = np.mean(z2xs)
```

```
z2y = np.mean(z2ys)
z2x, z2y
```

```
## (3.25, 8.0)
```



b.) Based on the centroids you found above reassign the memberships by using Manhattan distance.

```
xs = np.array([2,2,8,5,7,6,1,4])
ys = np.array([10,5,4,8,5,4,2,9])
z1x = 5.5
z1y = 3.75
z1x_diff = np.abs(z1x - xs)
z1y_diff = np.abs(z1y - ys)
z1_dist = z1x_diff + z1y_diff
z1_dist
```

```
## array([9.75, 4.75, 2.75, 4.75, 2.75, 0.75, 6.25, 6.75])
```

```
z2x = 3.25
z2y = 8
z2x_diff = np.abs(z2x - xs)
z2y_diff = np.abs(z2y - ys)
z2_dist = z2x_diff + z2y_diff
z2_dist
```

```
## array([3.25, 4.25, 8.75, 1.75, 6.75, 6.75, 8.25, 1.75])
```

p_i	a	b	c	d	e	f	g	h
d(p_i, z_1)	9.75	4.75	2.75	4.75	2.75	0.75	6.25	6.75
d(p_i, z_2)	3.25	4.25	8.75	1.75	6.75	6.75	8.25	1.75

We can now see that we have clusters $C_1 = \{c, e, f, g\}$ and $C_2 = \{a, b, d, h\}$. In fact, our clusters are unchanged.

5. Given the distance matrix below answer the following questions. Notice that this is a distance matrix, meaning the distance between any pair of points can be found by checking the corresponding cell to them. ***(20 points)***

---

	a	b	c	d	e	f	g	h
a	0	5	8	4	7	8	8	2
b	5	0	6	4	5	4	3	4
c	8	6	0	5	1	2	7	6
d	4	4	5	0	4	4	7	1
e	7	5	1	4	0	1	7	5
f	8	4	2	4	1	0	5	5
g	8	3	7	7	7	5	0	8
h	2	4	6	1	5	5	8	0

a.) Perform hierarchical clustering using *single link* measure for the above and draw the final dendrogram.

Itteration 1

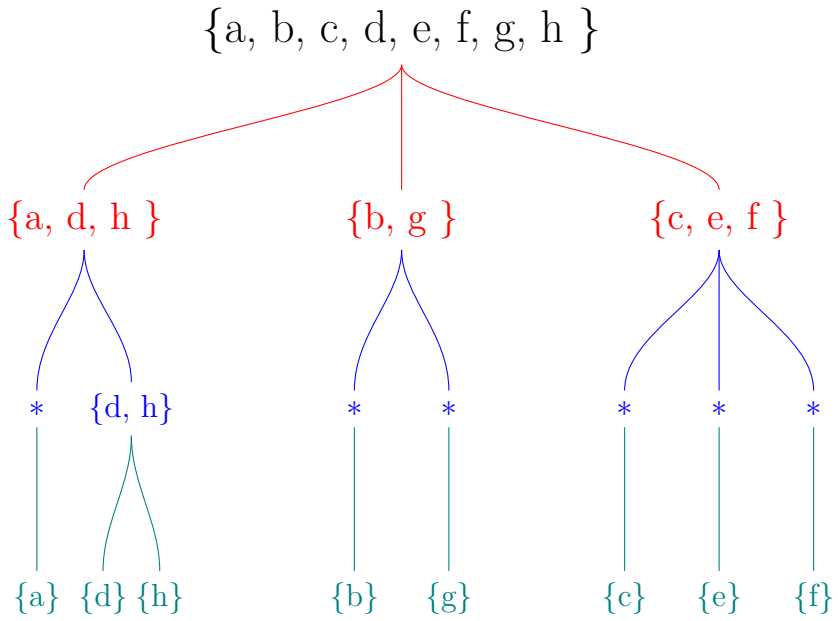
	a	b	c:e:f	d:h	g
a	0	5	7	2	8
b	5	0	4	4	3
c:e:f	7	4	0	4	5
d:h	2	4	4	0	7
g	8	3	5	7	0

Itteration 2

	a:d:h	b	c:e:f	g
a:d:h	0	4	4	7
b	4	0	4	3
c:e:f	4	4	0	5
g	7	3	5	0

Itteration 3

	a:d:h	b:g	c:e:f
a:d:h	0	4	4
b:g	4	0	4
c:e:f	4	4	0



b.) Determine whether a point is *core* based on $\epsilon = 6$ and $minPts = 2$.

To meet this requirement we must have 2 points within a distance of 6 for a point to be considered *core*.

	a	b	c	d	e	f	g	h
a	0	5	8	4	7	8	8	2
b	5	0	6	4	5	4	3	4
c	8	6	0	5	1	2	7	6
d	4	4	5	0	4	4	7	1
e	7	5	1	4	0	1	7	5
f	8	4	2	4	1	0	5	5
g	8	3	7	7	7	5	0	8
h	2	4	6	1	5	5	8	0

It is easy to see that under this requirement all points are *core*.