CS 484: Intro to Machine Learning

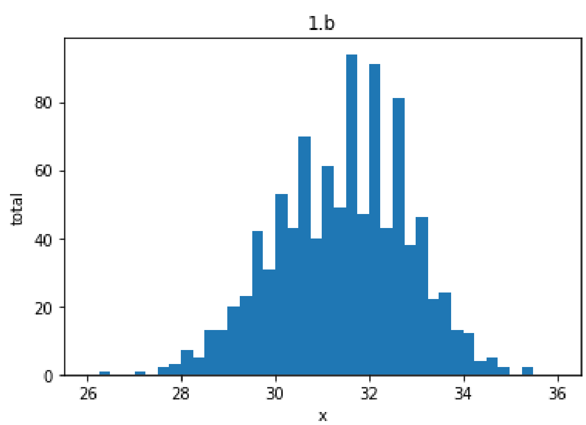
Fall 2020 Assignment 1

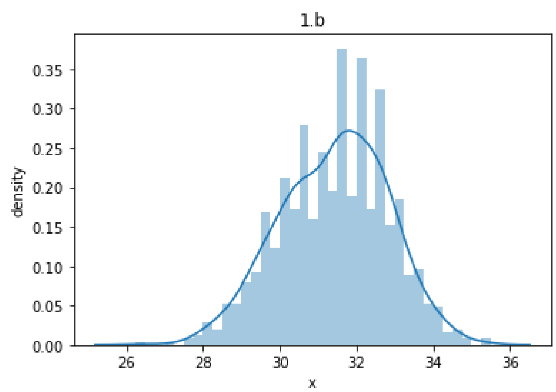
Question 1 (30 points)

Write a Python program to calculate the density estimator of a histogram. Use the field *x* in the NormalSample.csv file.

a) (5 points) Let a be the largest integer less than the minimum value of the field x, and b be the smallest integer greater than the maximum value of the field x. What are the values of a and b?
The minimum value of the field x is *26.3* and maximum value is *35.4*. Thus, a is *26* and b is *36*.

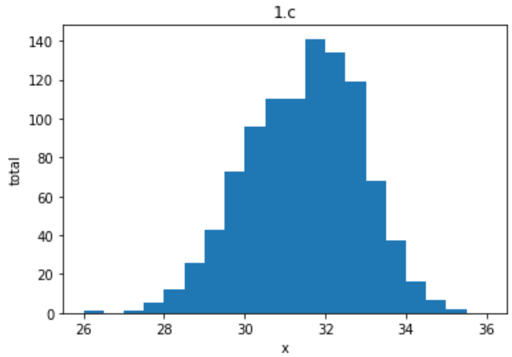
b) (5 points) Use h = 0.25, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.
[(26.125, 0.0), (26.375, 0.003996003996003996), (26.625, 0.0), (26.875, 0.0), (27.125, 0.003996003996003996), (27.375, 0.0), (27.625, 0.007992007992007992), (27.875, 0.011988011988011988), (28.125, 0.027972027972027972), (28.375, 0.01998001998001998), (28.625, 0.05194805194805195), (28.875, 0.05194805194805195), (29.125, 0.07992007992007992), (29.375, 0.0919080919080919), (29.625, 0.16783216783216784), (29.875, 0.12387612387612387), (30.125, 0.21178821178821178), (30.375, 0.17182817182817184), (30.625, 0.27972027972027974), (30.875, 0.15984015984015984), (31.125, 0.24375624375624375), (31.375, 0.1958041958041958), (31.625, 0.3756243756243756), (31.875, 0.1878121878121878), (32.125, 0.36363636363636365), (32.375, 0.17182817182817184), (32.625, 0.32367632367632365), (32.875, 0.15184815184815184), (33.125, 0.1838161838161838), (33.375, 0.08791208791208792), (33.625, 0.0959040959040959), (33.875, 0.05194805194805195), (34.125, 0.04795204795204795), (34.375, 0.015984015984015984), (34.625, 0.01998001998001998), (34.875, 0.007992007992007992), (35.125, 0.0), (35.375, 0.007992007992007992), (35.625, 0.0), (35.875, 0.0)]

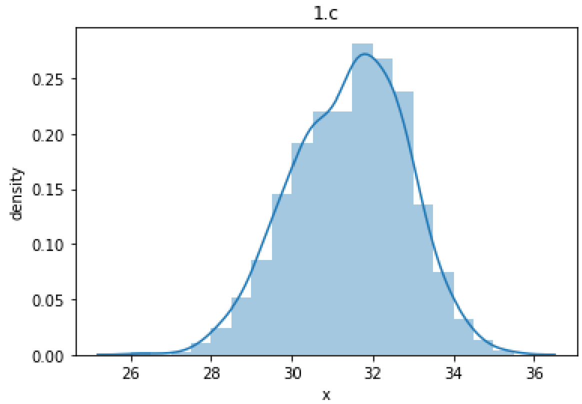




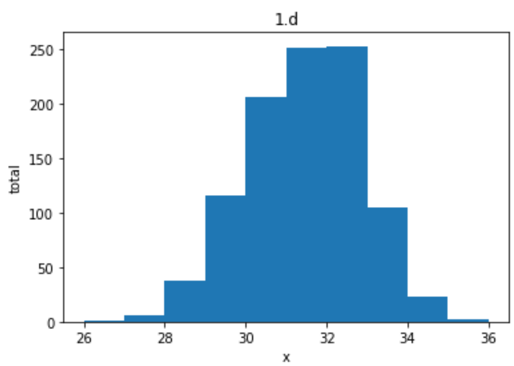
c) (5 points) Use h = 0.5, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.
[(26.25, 0.001998001998001998), (26.75, 0.0), (27.25, 0.001998001998001998), (27.75, 0.00999000999000999), (28.25, 0.023976023976023976), (28.75, 0.05194805194805195), (29.25, 0.08591408591408592), (29.75, 0.14585414585414586), (30.25, 0.1918081918081918), (30.75, 0.21978021978021978), (31.25, 0.21978021978021978), (31.75, 0.2817182817182817), (32.25, 0.2677322677322677), (32.75, 0.23776223776223776), (33.25, 0.13586413586413587), (33.75,

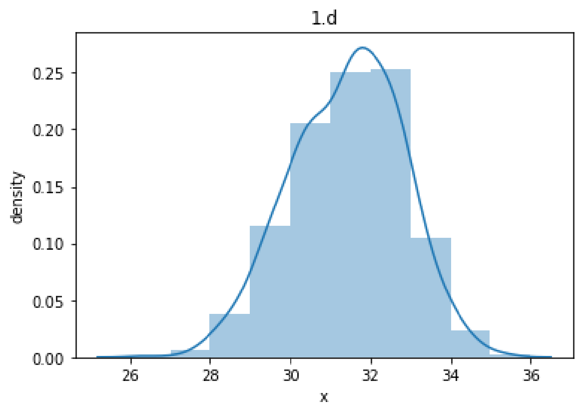
0.07392607392607392), (34.25, 0.03196803196803197), (34.75, 0.013986013986013986), (35.25, 0.003996003996003996), (35.75, 0.0)]



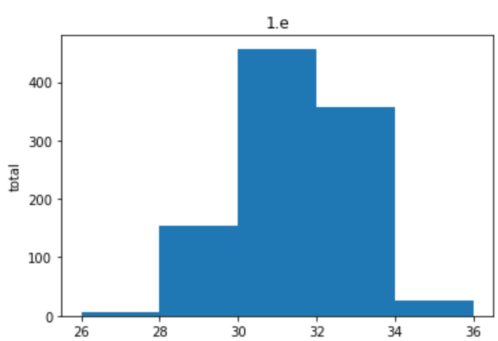


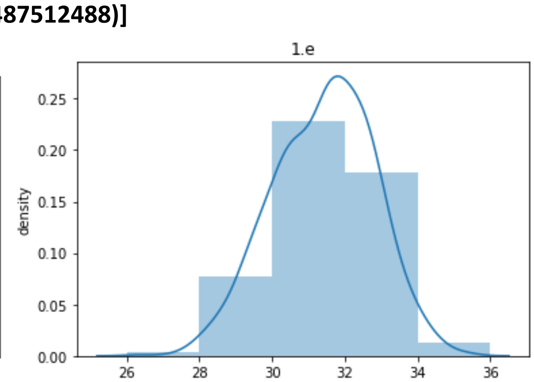
d) (5 points) Use h = 1, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.
[(26.5, 0.000999000999000999), (27.5, 0.005994005994005994), (28.5, 0.03796203796203796), (29.5, 0.11588411588411589), (30.5, 0.2057942057942058), (31.5, 0.25074925074925075), (32.5, 0.25274725274725274), (33.5, 0.1048951048951049), (34.5, 0.022977022977022976), (35.5, 0.001998001998001998)]





e) (5 points) Use h = 2, minimum = a and maximum = b. List the coordinates of the density estimator. Paste the histogram drawn using Python or your favorite graphing tools.
[(27.0, 0.0034965034965034965), (29.0, 0.07692307692307693), (31.0, 0.22827172827172826), (33.0, 0.17882117882117882), (35.0, 0.012487512487512488)]





f) (5 points) Among the four histograms, which one, in your honest opinions, can best provide your insights into the shape and the spread of the distribution of the field x? Please state your arguments.
I would say h = 0.5 because it is the most uniform, can see the shape and spread of the distribution better than other h's, and also best fits the probability density estimation in terms of the 'area' it covers.

Question 2 (10 points)

Use in the NormalSample.csv to generate box-plots for answering the following questions.

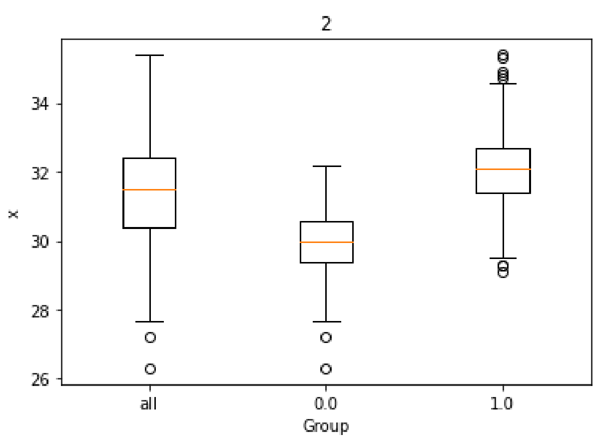
a) (5 points) What is the five-number summary of x for each category of the group? What are the values of the 1.5 IQR whiskers for each category of the group?

All x summary
Min: 26.3
Q1: 30.4
Median: 31.5
Q3: 32.4
Max: 35.4
1.5 IQR whiskers = (27.4, 35.4)

0.0 summary
Min: 26.3
Q1: 29.4
Median: 30.0
Q3: 30.6
Max: 35.4
1.5 IQR whiskers = (27.6, 32.4)

1.0 summary
Min: 26.3
Q1: 31.4
Median: 32.1
Q3: 32.7
Max: 35.4
1.5 IQR whiskers = (29.45, 34.65)

b) (5 points) Draw a graph where it contains the boxplot of x, the boxplot of x for each category of Group (i.e., three boxplots within the same graph frame). Use the 1.5 IQR whiskers, identify the outliers of x, if any, for the entire data and for each category of the group.



All : 2 outliers = [26.3 27.2]
0 : 2 outliers = [26.3 27.2]
1 : 8 outliers = [29.1 29.3 29.3 34.7 34.8 34.9 35.3 35.4]

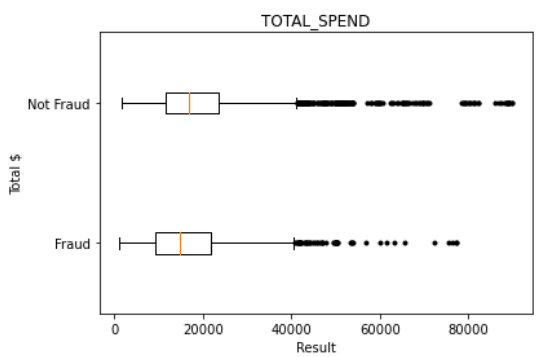
Question 3 (40 points)

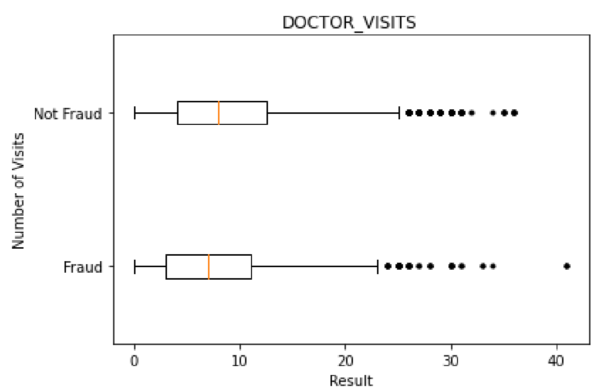
The data, FRAUD.csv, contains results of fraud investigations of 5,960 cases. The binary variable FRAUD indicates the result of a fraud investigation: 1 = Fraudulent, 0 = Otherwise. The other interval variables contain information about the cases.

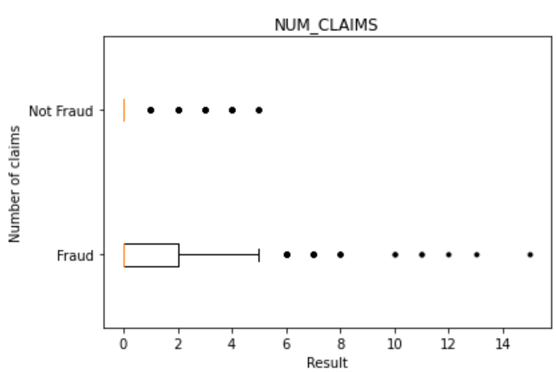
1. TOTAL_SPEND: Total amount of claims in dollars
2. DOCTOR_VISITS: Number of visits to a doctor
3. NUM_CLAIMS: Number of claims made recently
4. MEMBER_DURATION: Membership duration in number of months
5. OPTOM_PRESC: Number of optical examinations
6. NUM_MEMBERS: Number of members covered

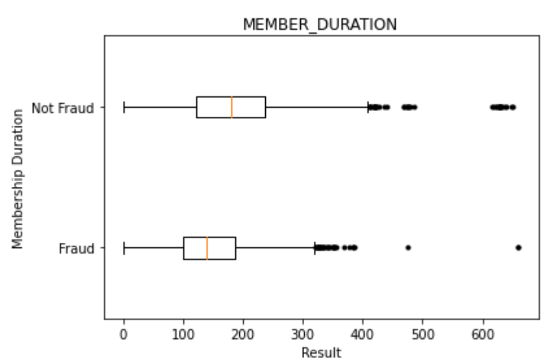
You are asked to use the Nearest Neighbors algorithm to predict the likelihood of fraud.

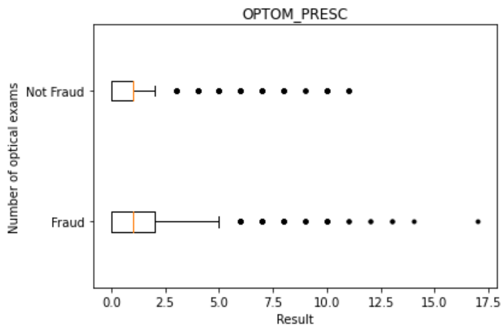
a) (5 points) What percent of investigations are found to be fraudulent? Please give your answer up to 4 decimal places.
   Total: 5960, Fraud: 1189, Not Fraud: 4771
   19.9496% of investigations are found to be fraudulent
b) (5 points) Use the BOXPLOT function to produce horizontal box-plots. For each interval variable, one box-plot for the fraudulent observations, and another box-plot for the non-fraudulent observations. These two box-plots must appear in the same graph for each interval variable.

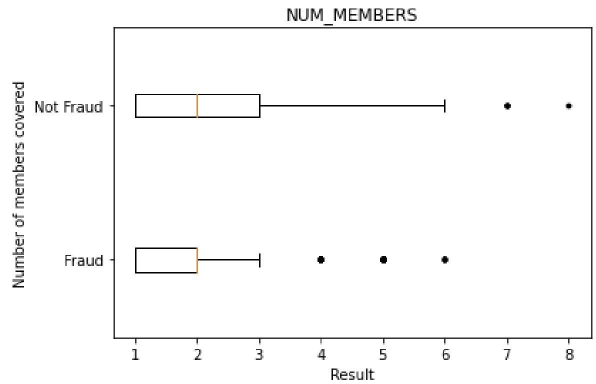












c) (10 points) Orthonormalize interval variables and use the resulting variables for the nearest neighbor analysis. Use only the dimensions whose corresponding eigenvalues are greater than one.

i. (5 points) How many dimensions are used?

6 dimensions are used

ii. (5 points) Please provide the transformation matrix? You must provide proof that the resulting variables are actually orthonormal.

Transformation matrix:

[[-6.49862374e-08 -2.41194689e-07 2.69941036e-07 -2.42525871e-07 -7.90492750e-07 5.96286732e-07]
[7.31656633e-05 -2.94741983e-04 9.48855536e-05 1.77761538e-03 3.51604254e-06 2.20559915e-10]
[-1.18697179e-02 1.70828329e-03 -7.68683456e-04 2.03673350e-05 1.76401304e-07 9.09938972e-12]
[1.92524315e-06 -5.37085514e-05 2.32038406e-05 -5.78327741e-05 1.08753133e-04 4.32672436e-09]
[8.34989734e-04 -2.29964514e-03 -7.25509934e-03 1.11508242e-05 2.39238772e-07 2.85768709e-11]
[2.10964750e-03 1.05319439e-02 -1.45669326e-03 4.85837631e-05 6.76601477e-07 4.66565230e-11]]

Transpose of transformed x multiplied by transformation matrix should result in identity matrix

```
transf_m = eigen_vec * np.linalg.inv(np.sqrt(np.diagflat(eigen_val)))
print('Transformation matrix :', transf_m)
# transformed x
transf_x = x * transf_m
# check if identity matrix
result = transf_x.transpose() * transf_x
#print(result)
```

Result is : [[1.00000000e+00 -6.28837260e-17 -1.69753534e-15 8.78069590e-15 1.00267017e-15 -2.18900419e-16]
[-6.28837260e-17 1.00000000e+00 -5.58580959e-16 -1.91886437e-14 -1.51354623e-15 -2.93168267e-16]
[-1.69753534e-15 -5.58580959e-16 1.00000000e+00 1.73819292e-15 8.15320034e-17 -2.94902991e-17]

**[8.78069590e-15 -1.91886437e-14 1.73819292e-15 1.00000000e+00
  1.13286117e-14 -3.82766735e-15]
 [1.00267017e-15 -1.51354623e-15 8.15320034e-17 1.13286117e-14
  1.00000000e+00 -6.27969898e-16]
 [-2.18900419e-16 -2.93168267e-16 -2.94902991e-17 -3.82766735e-15
 -6.27969898e-16 1.00000000e+00]]**

= Which is an identity matrix because the diagonal is 1, while others are basically 0

Also, orthonormalizing using the orth function from scipy and confirming:

```
orth_x = LA2.orth(x)
check = orth_x.transpose().dot(orth_x)
# should be identity matrix
print(check)
```

**[[1.00000000e+00 -7.97972799e-17 9.10729825e-17 -2.77555756e-17
 -2.42861287e-17 -4.33680869e-18]
 [-7.97972799e-17 1.00000000e+00 -1.90819582e-17 2.94902991e-17
 -5.20417043e-17 -6.14742632e-17]
 [9.10729825e-17 -1.90819582e-17 1.00000000e+00 8.45677695e-18
  8.13151629e-18 -3.16858085e-17]
 [-2.77555756e-17 2.94902991e-17 8.45677695e-18 1.00000000e+00
  3.26995375e-16 -8.51098705e-18]
 [-2.42861287e-17 -5.20417043e-17 8.13151629e-18 3.26995375e-16
  1.00000000e+00 3.38921599e-16]
 [-4.33680869e-18 -6.14742632e-17 -3.16858085e-17 -8.51098705e-18
  3.38921599e-16 1.00000000e+00]]**

You also get an identity matrix,
So the resulting variables are actually orthonormal

d) (10 points) Use the NearestNeighbors module to execute the Nearest Neighbors algorithm using exactly <u>five</u> neighbors and the resulting variables you have chosen in c). The KNeighborsClassifier module has a score function.

   i. (5 points) Run the score function, provide the function return value
   0.8778523489932886

   ii. (5 points) Explain the meaning of the score function return value.
   The value represents the accuracy of the function with the given data, meaning the model has a 87.78% accuracy.

e) (5 points) For the observation which has these input variable values: TOTAL_SPEND = 7500, DOCTOR_VISITS = 15, NUM_CLAIMS = 3, MEMBER_DURATION = 127, OPTOM_PRESC = 2, and NUM_MEMBERS = 2, find its **five** neighbors. Please list their input variable values and the target values. *Reminder: transform the input observation using the results in c) before finding the neighbors*.

```
# observation
obsv = [[7500, 15, 3, 127, 2, 2]]
transf_obsv = obsv * transf_m
obsv_neigh = neighbors.kneighbors(transf_obsv)
# print the distance and the element # of the neighbors
obsv_neighbors = [(distance, element)
                  for distance, element in zip(list(obsv_neigh[0][0]),
                                               list(obsv_neigh[1][0]))]
print(obsv_neighbors)
# print the neighbors values
for neighb in obsv_neighbors:
    print(data[neighb[1]])

# print probability of fraud
print(neighbors.predict_proba(transf_obsv))
```

5 neighbors - Tuple of distance and element of neighbors = [(6.585445079827193e-10, 588), (0.010457155671774961, 2897), (0.01206994606792243, 1199), (0.012186196058304651, 1246), (0.014037896643257355, 886)]

Values of the 5 neighbors:

{'CASE_ID': 589.0, 'FRAUD': 1.0, 'TOTAL_SPEND': 7500.0, 'DOCTOR_VISITS': 15.0, 'NUM_CLAIMS': 3.0, 'MEMBER_DURATION': 127.0, 'OPTOM_PRESC': 2.0, 'NUM_MEMBERS': 2.0}

{'CASE_ID': 2898.0, 'FRAUD': 1.0, 'TOTAL_SPEND': 16000.0, 'DOCTOR_VISITS': 18.0, 'NUM_CLAIMS': 3.0, 'MEMBER_DURATION': 146.0, 'OPTOM_PRESC': 3.0, 'NUM_MEMBERS': 2.0}

{'CASE_ID': 1200.0, 'FRAUD': 1.0, 'TOTAL_SPEND': 10000.0, 'DOCTOR_VISITS': 16.0, 'NUM_CLAIMS': 3.0, 'MEMBER_DURATION': 124.0, 'OPTOM_PRESC': 2.0, 'NUM_MEMBERS': 1.0}

{'CASE_ID': 1247.0, 'FRAUD': 1.0, 'TOTAL_SPEND': 10200.0, 'DOCTOR_VISITS': 13.0, 'NUM_CLAIMS': 3.0, 'MEMBER_DURATION': 119.0, 'OPTOM_PRESC': 2.0, 'NUM_MEMBERS': 3.0}

{'CASE_ID': 887.0, 'FRAUD': 1.0, 'TOTAL_SPEND': 8900.0, 'DOCTOR_VISITS': 22.0, 'NUM_CLAIMS': 3.0, 'MEMBER_DURATION': 166.0, 'OPTOM_PRESC': 1.0, 'NUM_MEMBERS': 2.0}

f) (5 points) Follow-up with e), what is the predicted probability of fraudulent (i.e., FRAUD = 1)? If your predicted probability is greater than or equal to your answer in a), then the observation will be classified as fraudulent. Otherwise, non-fraudulent. Based on this criterion, will this observation be misclassified?

When I do the function predict_proba, it results in [[0, 1.]] which means 0% chance its not fraud, and 100% its fraud. Since the predicted probability is greater than the answer in a, 19%, the observation is classified to be fraudulent and probably wouldn't be misclassified as it has a really high probability, 100%.

Question 4 (10 points)

Try the Manhattan distance and the Chebyshev distance in analyzing the ten-observations example in Chapter 2 of *A Practitioner's Guide to Machine Learning*. Does the optimal number of neighbors the same as that found with the Euclidean distance?

Yes, optimal number of neighbors is the same as found with Euclidean, 2 neighbors

Using python, redid the steps for Manhattan and Chebyshev and got that 2 neighbors is the most optimal.

```python
# euclidean = math.sqrt((current[0] - check[0]) ** 2 + (current[1] - check[1]) ** 2)
# manhattan = abs(current[0] - check[0]) + abs(current[1] - check[1])
# chebyshev = max(abs(current[0] - check[0]), abs(current[1] - check[1]))
for i_nbr in range(1, 10):
    k = i_nbr
    step1 = []
    i = 0
    for current in data:
        step1.append([])
        step1[i] = []
        for check in data:
            # replace distance for different stuff
            distance = math.sqrt((current[0] - check[0]) ** 2 + (current[1] - check[1]) ** 2)
            step1[i].append(np.round(distance, decimals = 4))
        i += 1
    #pprint(step1)
    step2 = []
    for row in step1:
        array = np.array(row)
        temp = array.argsort()
        ranks = np.empty_like(temp)
        ranks[temp] = np.arange(len(array)) + 1
        step2.append(list(ranks))
    #pprint(step2)
    step3 = []
    for i in range(len(step2)):
        nbrs_y = [data[x][2] for x in range(len(step2)) if step2[i][x] <= k]
        y_guess = sum(nbrs_y) / len(nbrs_y)
        error = np.round(data[i][2] - y_guess, decimals = 3)
        squared = np.round(error ** 2, decimals = 3)
        #print(data[i][2], y_guess, error, squared)
        step3.append([data[i][2], y_guess, error, squared])
    #pprint(step3)
    sqr_err = [x[3] for x in step3]
    root_avg_sqr_err = math.sqrt(sum(sqr_err) / len(sqr_err))
    print(k, root_avg_sqr_err)
```

Manhattan :

```
1 0.0
2 0.7071067811865476
3 0.7817928114276825
4 0.8837420438114281
5 0.9612491872558333
6 0.9310209449845905
7 0.9783659846908005
8 0.9961425600786264
9 1.075499883774982
```

Chebyshev:

```
1 0.0
2 0.7071067811865476
3 0.8028698524667619
4 0.8475848040166837
5 0.9099450532861861
6 0.8930285549745877
7 0.9783659846908005
8 0.9898484732523457
9 1.075499883774982
```

Euclidean :

```
1 0.0
2 0.7071067811865476
3 0.8028698524667619
4 0.8586035173466273
5 0.9099450532861861
6 0.8930285549745877
7 0.9783659846908005
8 0.9898484732523457
9 1.075499883774982
```

Question 5 (10 points)

Include *Entropy_Image* in analyzing the banknote data mentioned in Chapter 2 of *A Practitioner's Guide to Machine Learning*. Does orthonormalizing the four input features make a difference in determining the number of neighbors?

The result of the original and orthonormalizing with Entropy Image:

```
Original
      k  Misclassification Rate
0   1.0                0.000000
1   2.0                0.000000
2   3.0                0.000729
3   4.0                0.000000
4   5.0                0.000000
5   6.0                0.000000
6   7.0                0.000000
7   8.0                0.000000
8   9.0                0.000000
9  10.0                0.000000
--------------------
Orthonormalize
      k  Misclassification Rate
0   1.0                0.000000
1   2.0                0.000000
2   3.0                0.000000
3   4.0                0.000000
4   5.0                0.000000
5   6.0                0.000000
6   7.0                0.002187
7   8.0                0.000000
8   9.0                0.002187
9  10.0                0.002187
```

With the original, only misclassification rate at k = 3, but orthonormalizing, there are misclassification rates at k = 6, 8, 9.

So, orthonormalizing the four input features does make a difference in determining the number of neighbors.

```python
inputVar = ['Variance_Image', 'Skewness_Image', 'Kurtosis_Image', 'Entropy_image']
targetVar = 'Authenticity'
bankNote = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_
                        sep = ',', header = None)
bankNote = bankNote.rename(columns = {0: inputVar[0], 1: inputVar[1], 2: inputVar[2],
                                      3: inputVar[3], 4: targetVar})
X = bankNote[inputVar[0:4]]
y = bankNote[targetVar]
nObs = len(y)
misClassRate_orig = pd.DataFrame(columns = ['k', 'Misclassification Rate'])
kMax = 10
for i in range(kMax):
    k = i +1
    nbrs = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', metric = 'euclidean')
    model = nbrs.fit(X, y)
    yPredictProb = model.predict_proba(X)
    yPredict = min(y) + np.argmax(yPredictProb, axis = 1)
    MCE = np.sum(np.where(y == yPredict, 0 , 1)) / nObs
    misClassRate_orig.loc[i] = [k, MCE]
# without orthonormalizing
print('Original')
print(misClassRate_orig)
print('--------------------')
# orthonormalizing
X = np.matrix(bankNote[inputVar[0:4]].values)
xtx = X.transpose() * X
evals, evecs = np.linalg.eigh(xtx)
transf = evecs * np.linalg.inv(np.sqrt(np.diagflat(evals)))
transf_x = X * transf
misClassRate_orth = pd.DataFrame(columns = ['k', 'Misclassification Rate'])
for i in range(kMax):
    k = i + 1
    nbrs = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', metric = 'euclidean')
    model = nbrs.fit(transf_x, y)
    yPredictProb = model.predict_proba(transf_x)
    yPredict = min(y) + np.argmax(yPredictProb, axis = 1)
    MCE = np.sum(np.where(y == yPredict, 0 , 1)) / nObs
    misClassRate_orth.loc[i] = [k, MCE]
print('Orthonormalize')
print(misClassRate_orth)
```