COMP3340 Assignment 1 part 2

By Andrew cox - c3356087

# Question 1

## A).

As shown below some variables have an immensely strong correlation as seen in figure 5 which is the connection between raising one’s hand and visiting the resources. Other trends like this can also be seen in other places such as how girls often preform better than boys however these other connection are weaker than that between raising one’s hand and visiting the resources.

## B).

1. Students who raided their hands are more actively involved in study related works. ()

2. There is no apparent gender bias in topic/subject selection. ()

3. Girls seem to have better overall performance than boys. (T)

* According to the averages, Girls do on average have a higher performance than boys however it is marginal having a difference of 0.07 the full averages are below.

The average performance for Boys: 2.501639344262295

The average performance for Girls: 2.5714285714285716

4. Boys are generally more open to: discussion, visiting resources, and raising hands. (**F**)

* As seen in figures 1 – 3 there is verry little correlation between the above-mentioned attributes. Boys tends not be no corrilation however when using pearsons correlation we see that there is a trend twords boys participating in more discussions and visiting more resources:
  + Pearsons correlation between raising hands and and participating in discussions amoung boys: 0.40727
  + Pearsons correlation between raising hands and and participating in discussions amoung girls: -0.10125
  + Pearsons correlation between rasing hands and visiting resources amoung boys: 1.00000
  + Pearsons correlation between rasing hands and visiting resources amoung girls: 1.00000
  + Pearsons correlation between discussion and visiting resources amoung boys: 0.40727
  + Pearsons correlation between discussion and visiting resources amoung girls: -0.10125

5. Those who participated more usually perform better. (T)

* As see in figures 7 through 9 there is a trend toward those who rase their hand and participate in discussions to achieve higher marks, even if the correlation is slim.

Graphs

figure 1

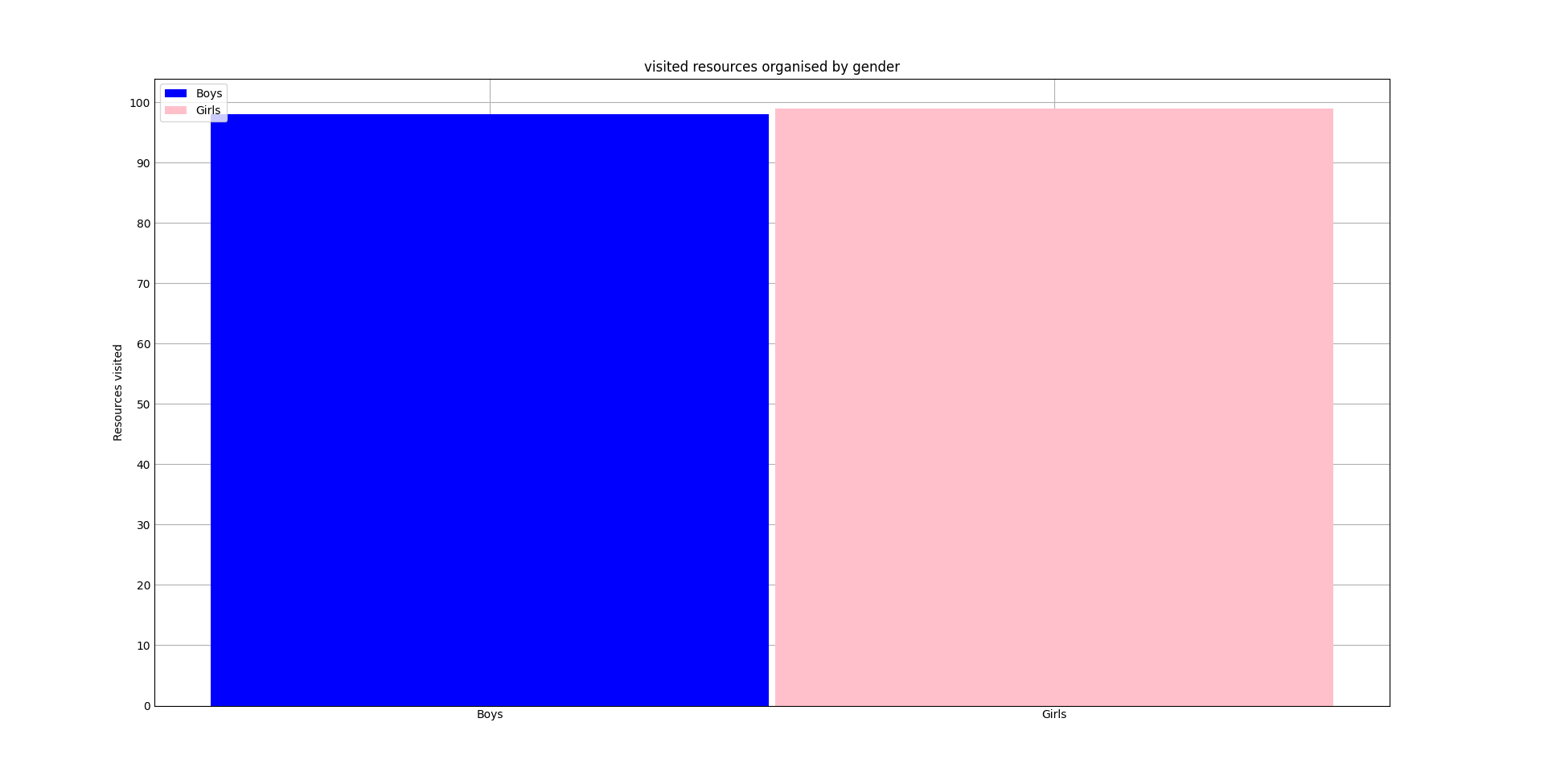
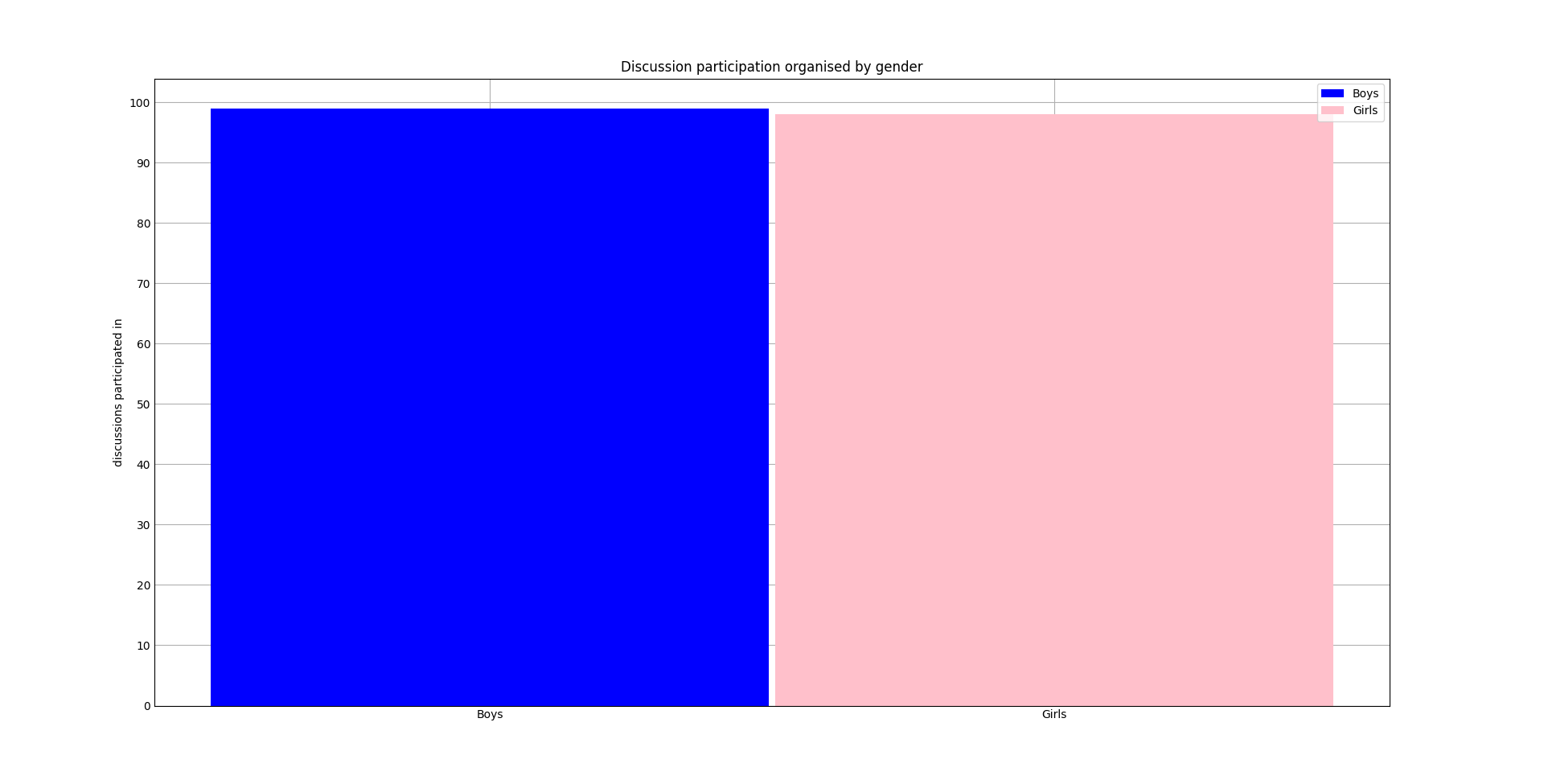
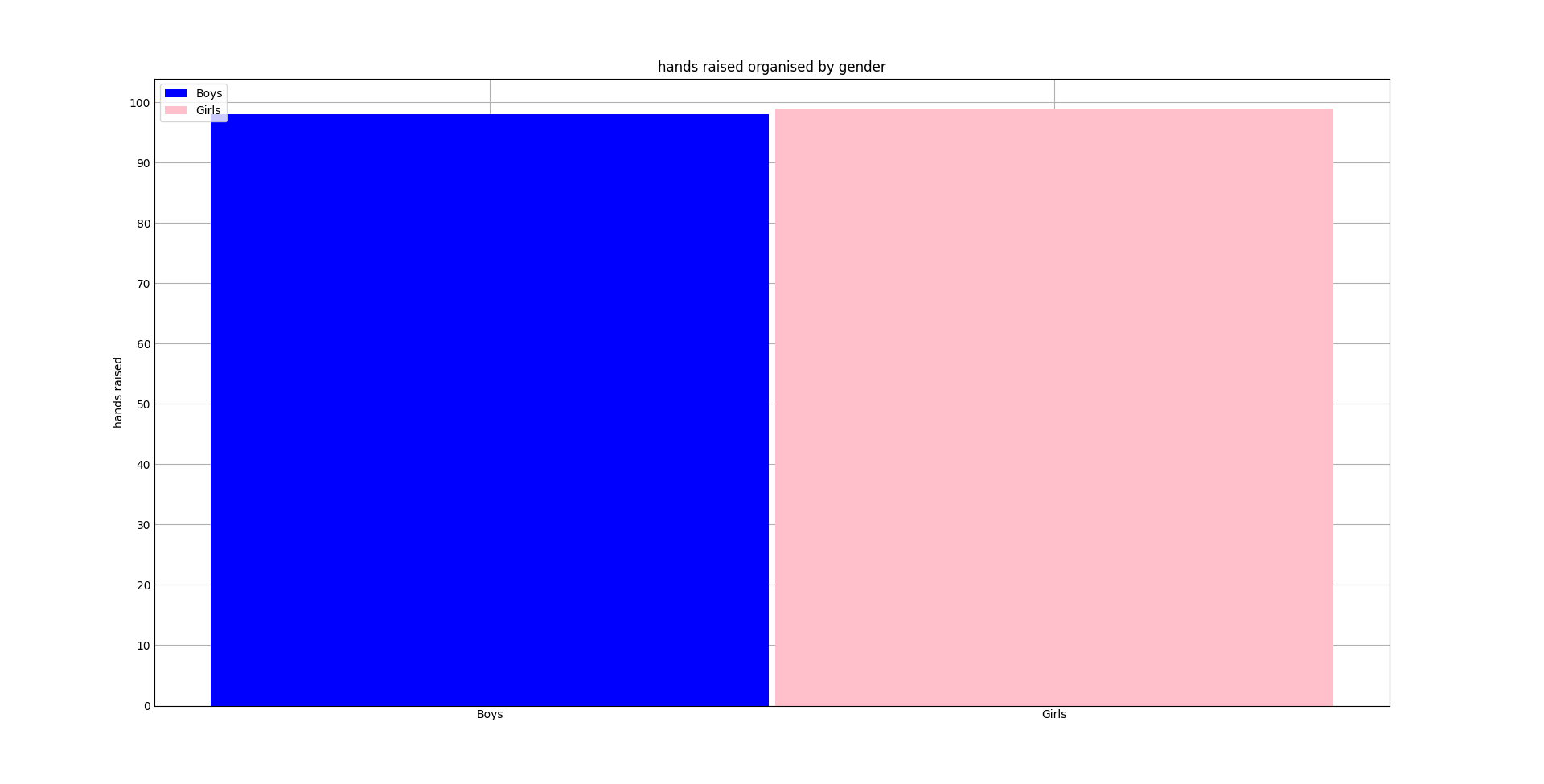
 figure2figure 3

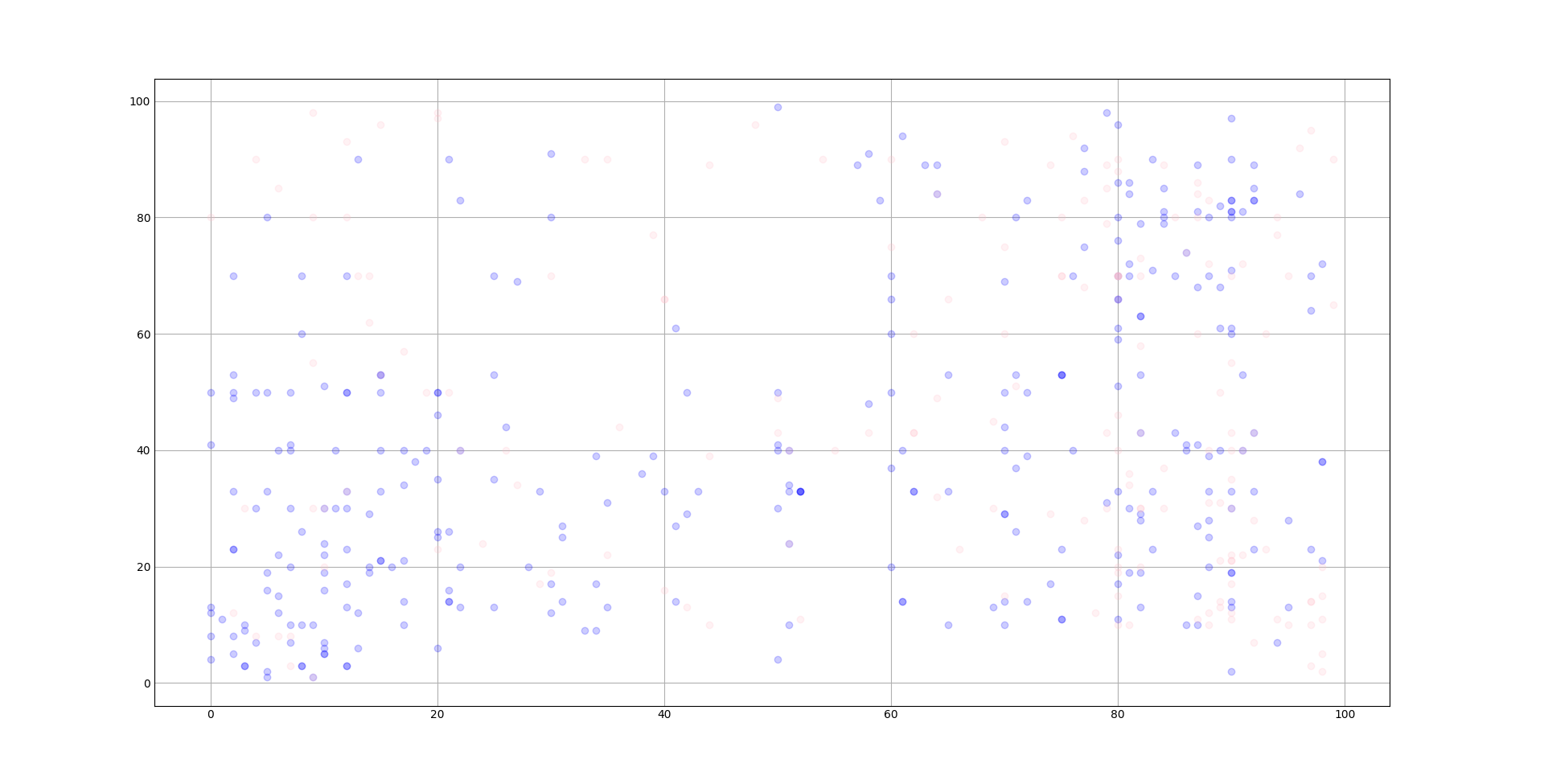
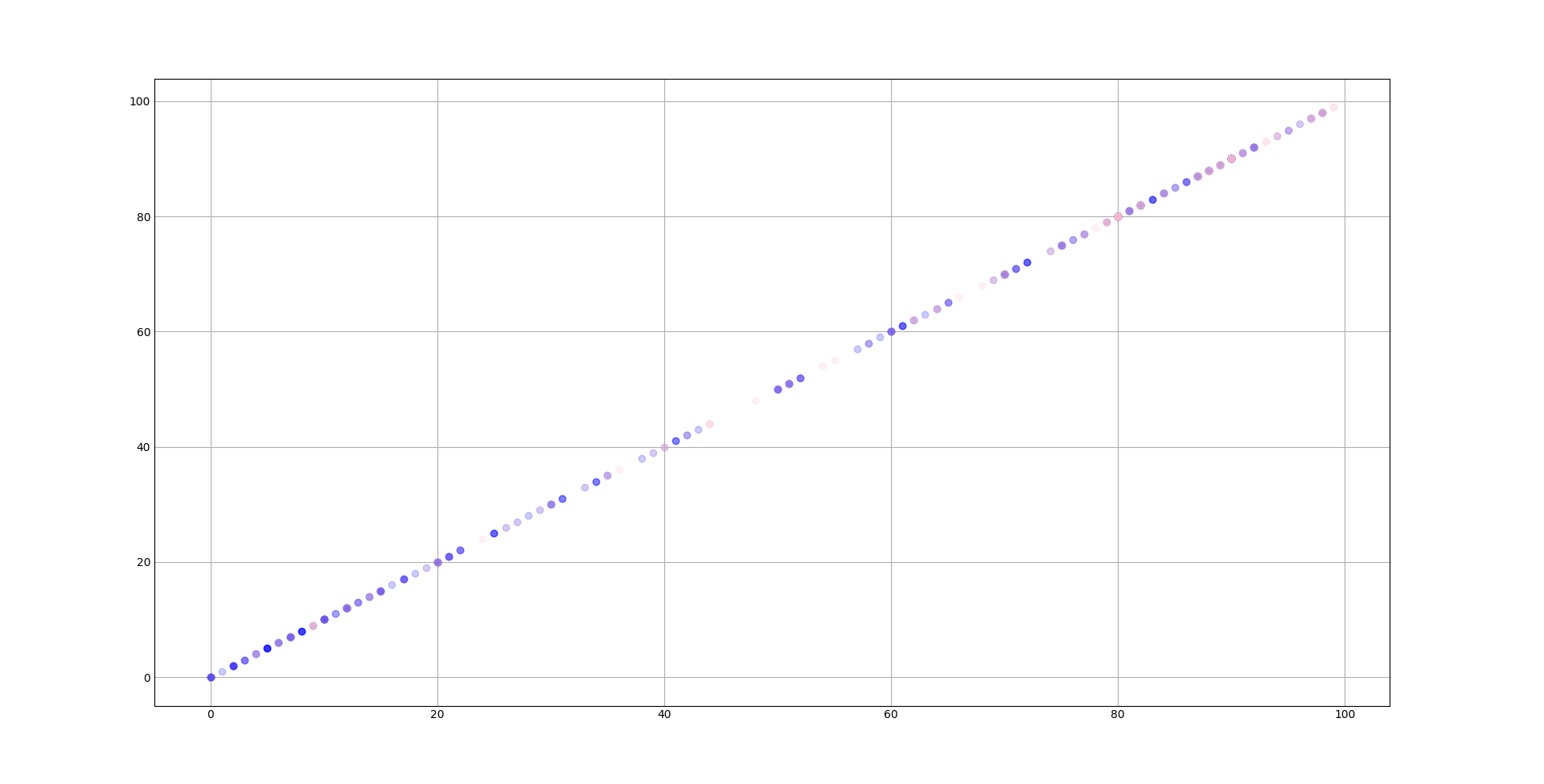
Figure 4 (x – raising hands, Y- discussion, Blue: boys, Pink: Girls)figure 5 (X - rasing ones hand, Y - visiting resources, Blue: boys, Pink: Girls)

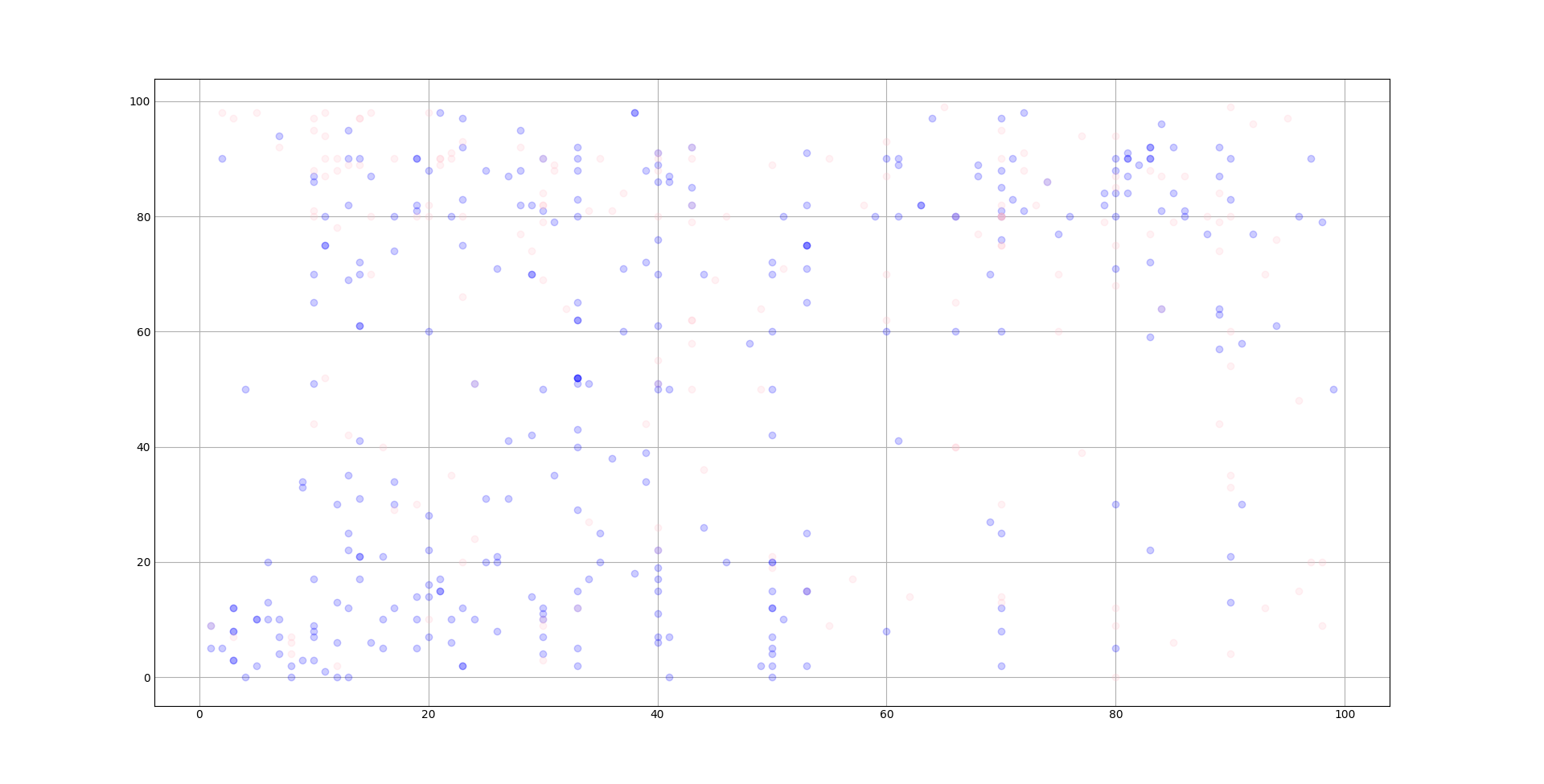
Figure 6 (X - Discussion participation, Y - visiting resources, Blue: boys, Pink: Girls) 

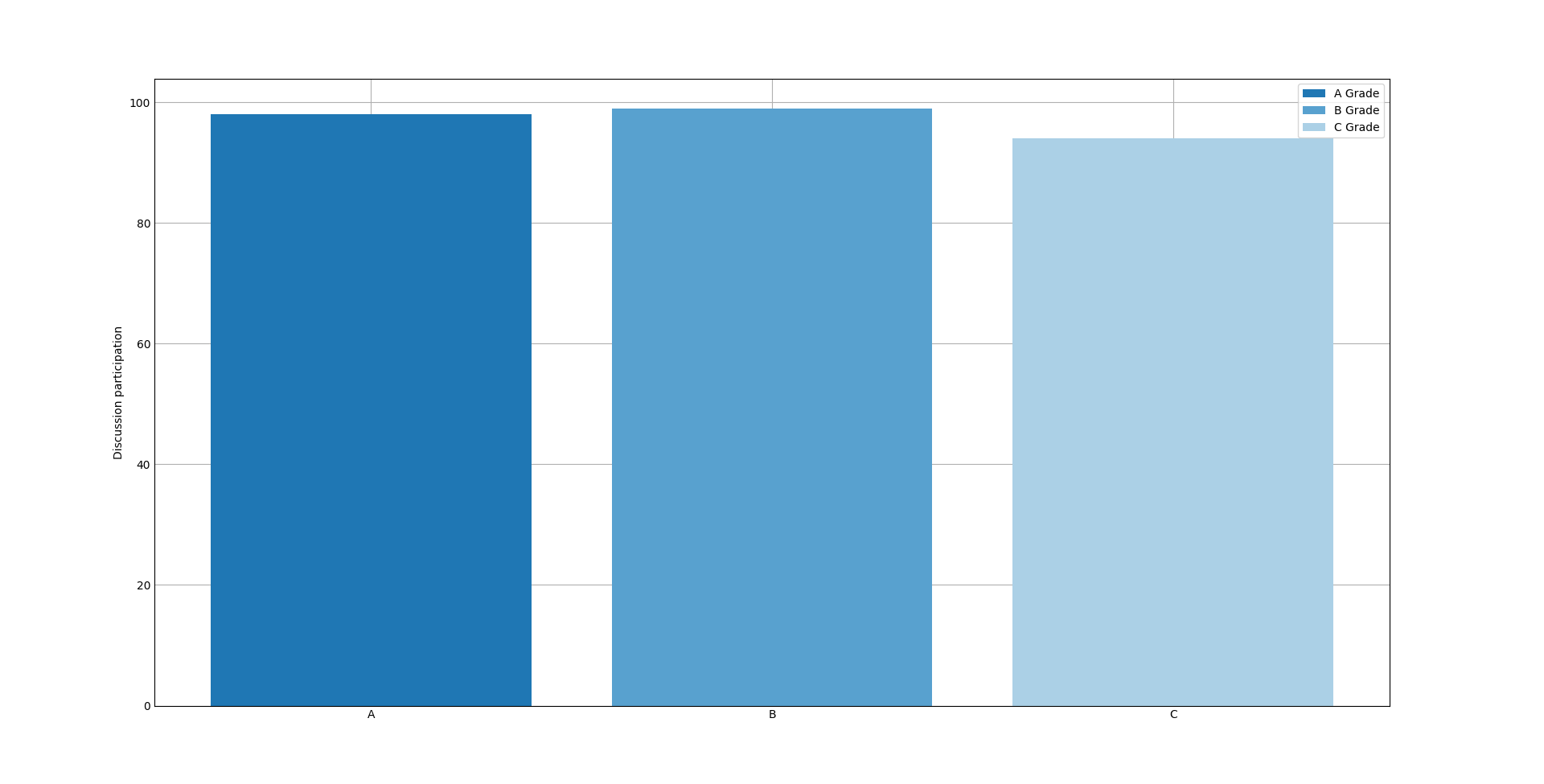
Figure 7

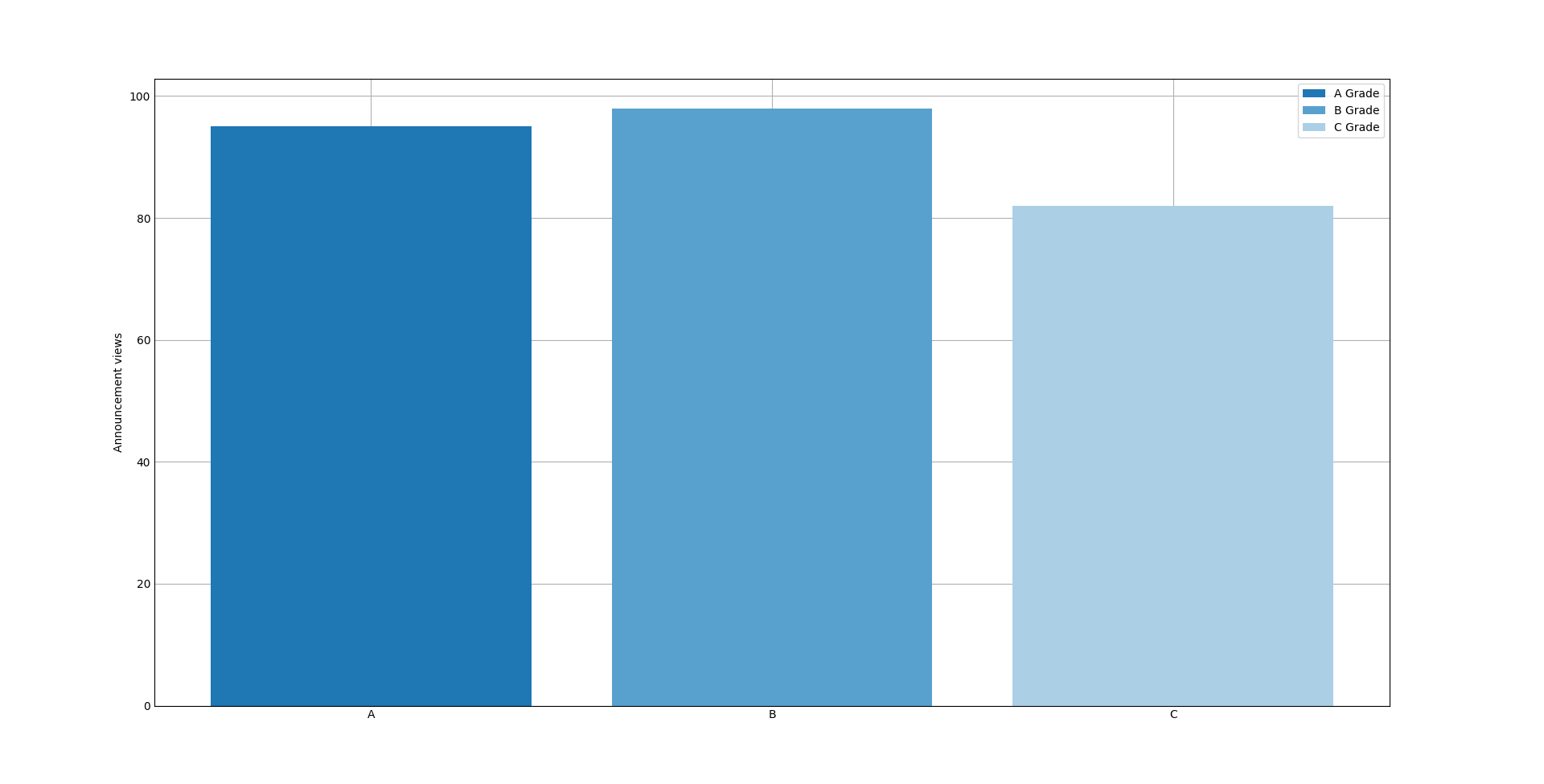
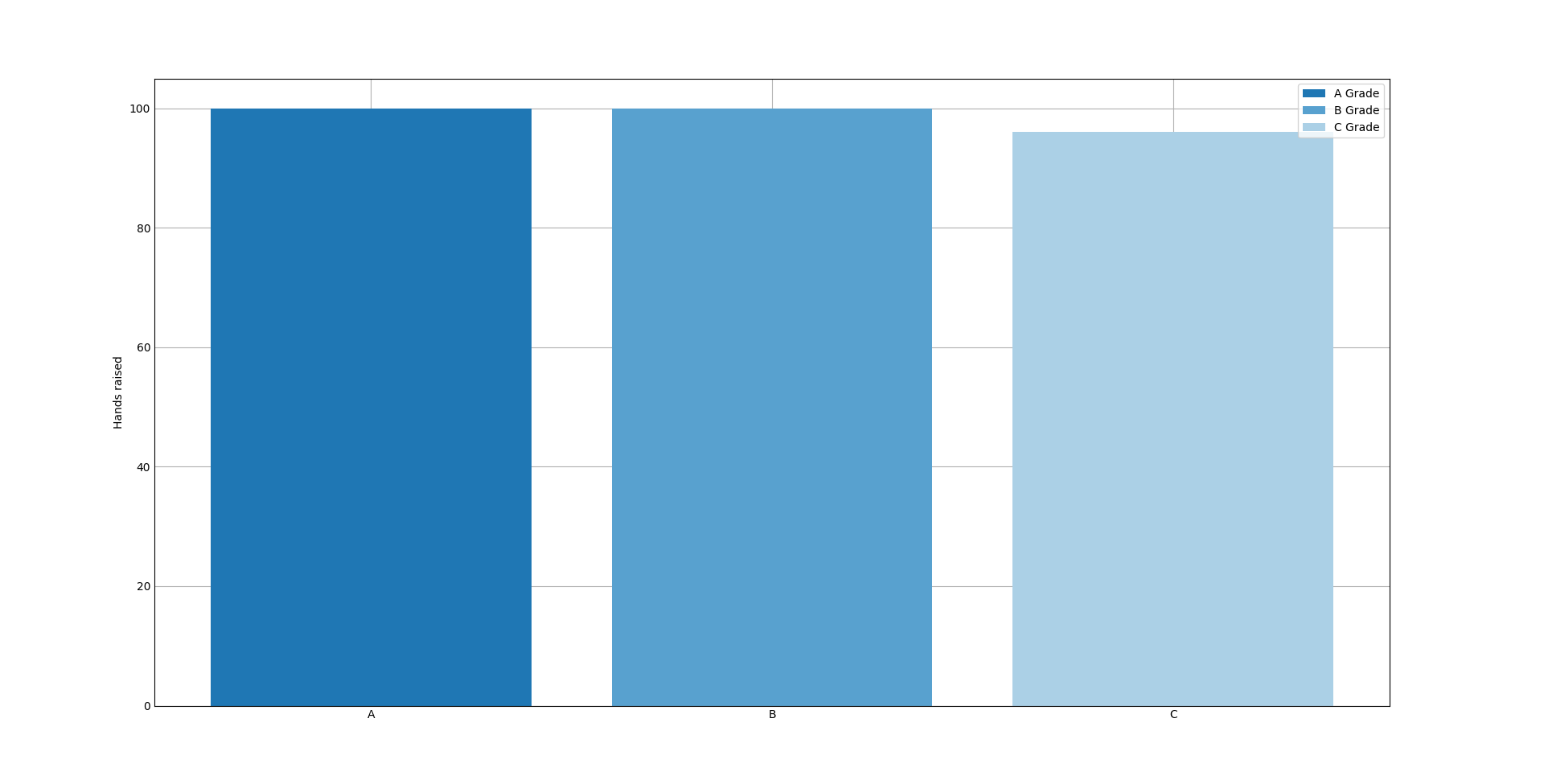
Figure 8

Figure 9

# Question 2

## Graphs

Proteins Relative neighbourhood graph



Samples Relative neighbourhood graph

Samples Minimum spanning tree.



Proteins Minimum spanning tree.



## Matrix’s

Proteins MST & RNG

Minimum spanning tree



Relative Neighbourhood Graph



Samples MST & RNG

Minimum spanning tree



Relative neighbourhood graph



## How was it done?

To generate the distance matrix I used the SCIPY euclidean() method. This checks the distance between two input arrays no matter of their contents. Where as with the hemming matrix it would only work properly for binary. I just used the same method I created in assignment 1 part 1 for question 1 retrofitted to generate a Euclidian matrix rather than a hemming matrix. For the samples I transposed the matrix using the numpy transpose() method.

To Actually generate the MST and RNG I used external libraries from SCIPY and relativeNeighborhoodGraph respectively. To generate the MST I parsed the distance matrix, index names and the export location to the generic method genMST() which used the SCIPY method minimum\_spanning\_tree() to generate a minimum spanning tree using Kruskal’s algorithm. The minimum\_spanning\_tree() method would return a sparse array which was then to be converted into a normal array. And exported to an XLSX file using pandas.  
To generate the RNG I used the same method as with he MST however with another step. The method returnRNG() returns a pandas data frame with duplicated lines, to fix this issue I created a method to loop through the matrix checking for duplicate lines (a line is duplicate if it appears in both x, y and y, x).

# Question 3

Results:

== Results for the test set ==

Accuracy: 0.8148148148148148

Mathews Correlation: 0.6342269356257724

F-1 Score: 0.8148148148148148

Specificity: 0.8536585365853658

Sensitivity: 0.7948717948717948

== Results for the test MCI set ==

Accuracy: 0.9090909090909091

Mathews Correlation: 0.0

F-1 Score: 0.9090909090909091

Specificity: 0.9090909090909091

Sensitivity: nan

How I did Section (a):  
I loaded the data into the program using pandas taking the data with one pass then taking the column names and row names with another. These where then split and flattened into useable arrays. To train the model I used sklearn’s SelectKBest() function along with the chi-squared (chi2) method to generate the kfeature set. However to use the chi2 method the data needs to be all positive, this is not true with the current data set. To remedy this issue I used sklearn’s preprocessing minmaxscalar() method to normalise the data. This normalised data is then fed into the selectKBest features using the fit\_transform() method to train the model.

How I did Section (b):  
To classify the data I defined the model using the KNeighborsClassifier() defining the n\_neighbors as 5. The model was then parsed the generated best features and the names of the samples(AD or NDC).

How I did Section (c):  
To complete section C I took the data from the different pages(Test Set AD and Test Set MCI) transposed them, Took the names of the samples to use as targets, and fed them into the model. The results are listed above. The only note worthy occurrences in this process was Test Set MCI had 2 rows of names and variables I removed the top one as it was not applicable to this process and there where samples that the model was not trained to recognised, they where not included in the predictions.

To gather the above information I used a few method from sklearn, namely f1\_score() to generate the f1-score, I used the setting “Micro” setting as this method seemed the most appropriate, it takes into account the True negatives, False negatives, and False positives. For Mathews correlation I used the Matthews\_corrcoef() method in sklearn, I takes the true values and the predicted values as arrays and returns the correlation coefficient. And the accuracy was calculated using the accuracy\_score() method in sklearn it works the same as the Mathews correlation coefficient.

To calculate the Specificity and Sensitivity, I used another sklearn method called confusion\_matrix(), this would return 4 variables, the True Negative rate, the True Positive rate, the False Positive rate, and the False Negative rate. This method took the true values, and the predicted values along with the labels that would appear in both variables. This when returned the varables mentioned above then those variables where plugged into their respective formula.

# Question 4

# Question 5

# Code

Question 1:

import matplotlib  
  
matplotlib.use('TkAgg')  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import scipy as sp  
  
  
def boop():  
 # changeing the data set for ease of use  
 data = pd.read\_csv("Datasets/xAPI-Edu-Data.csv")  
 dummy = pd.get\_dummies(data['gender'])  
 # modifying for male and female  
 data2 = pd.concat((data, dummy), axis=1)  
 data2 = data2.drop(['gender'], axis=1)  
 data2 = data2.drop(['M'], axis=1)  
 data2 = data2.rename(columns={"F": "Gender"})  
 # Female: 1 | Male: 0  
 dummy = pd.get\_dummies(data2['Relation'])  
 data2 = pd.concat((data2, dummy), axis=1)  
 data2 = data2.drop(['Relation'], axis=1)  
 data2 = data2.drop(['Father'], axis=1)  
 data2 = data2.rename(columns={"Mum": "Relation"})  
 # raised by the mother: 1 | raised by the father: 0  
  
 # ==== Graphing ==== #  
 plt.style.use('\_mpl-gallery')  
 # discussion  
 girlsdiscX = data2.loc[data2['Gender'] == 1]  
 girlsdsicY = girlsdiscX['Discussion'].to\_numpy()  
 girlsdiscX = girlsdiscX['Gender'].to\_numpy()  
  
 boysdiscX = data2.loc[data2['Gender'] == 0]  
 boysdiscY = boysdiscX['Discussion'].to\_numpy()  
 boysdiscX = boysdiscX['Gender'].to\_numpy()  
  
 # visiting resources  
 girlsvisX = data2.loc[data2['Gender'] == 1]  
 girlsvisY = girlsvisX['VisITedResources'].to\_numpy()  
 girlsvisX = girlsvisX['Gender'].to\_numpy()  
  
 boysvisX = data2.loc[data2['Gender'] == 0]  
 boysvisY = boysvisX['VisITedResources'].to\_numpy()  
 boysvisX = boysvisX['Gender'].to\_numpy()  
  
 # raising hands  
 girlrshndX = data2.loc[data2['Gender'] == 1]  
 girlrshndY = girlrshndX['VisITedResources'].to\_numpy()  
 girlrshndX = girlrshndX['Gender'].to\_numpy()  
  
 boyrshndX = data2.loc[data2['Gender'] == 0]  
 boyrshndY = boyrshndX['VisITedResources'].to\_numpy()  
 boyrshndX = boyrshndX['Gender'].to\_numpy()  
  
 # == Graph 1 == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.set\_title('visited resources organised by gender')  
 ax.set\_yticks(np.arange(0, 151, 10))  
 ax.set\_xticks(np.arange(2), ('Boys', 'Girls'))  
 ax.bar(boysvisX, boysvisY, color='blue', width=0.99, label='Boys')  
 ax.bar(girlsvisX, girlsvisY, color='pink', width=0.99, label='Girls')  
 # ax.bar(girlsdiscX, girlsdsicY, color='magenta', width=0.25, label='Female')  
 # ax.bar\_label(group1, padding=3)  
 ax.legend(loc='best')  
 ax.set\_ylabel("Resources visited")  
 # plt.xlabel(["Male", "Female"])  
 # plt.show()  
  
 # == Graph 2 == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.set\_title('Discussion participation organised by gender')  
 ax.set\_yticks(np.arange(0, 151, 10))  
 ax.set\_xticks(np.arange(2), ('Boys', 'Girls'))  
 ax.bar(boysdiscX, boysdiscY, color='blue', width=0.99, label='Boys')  
 ax.bar(girlsdiscX, girlsdsicY, color='pink', width=0.99, label='Girls')  
 # ax.bar(girlsdiscX, girlsdsicY, color='magenta', width=0.25, label='Female')  
 # ax.bar\_label(group1, padding=3)  
 ax.legend(loc='best')  
 ax.set\_ylabel("discussions participated in")  
 # ax.set\_yticklabels(["Male", "Female"])  
 # plt.xlabel(["Male", "Female"])  
 # plt.show()  
  
 # == Graph 3 == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.set\_title('hands raised organised by gender')  
 ax.set\_yticks(np.arange(0, 151, 10))  
 ax.set\_xticks(np.arange(2), ('Boys', 'Girls'))  
 ax.bar(boyrshndX, boyrshndY, color='blue', width=0.99, label='Boys')  
 ax.bar(girlrshndX, girlrshndY, color='pink', width=0.99, label='Girls')  
 # ax.bar(girlsdiscX, girlsdsicY, color='magenta', width=0.25, label='Female')  
 # ax.bar\_label(group1, padding=3)  
 ax.legend(loc='best')  
 # ax.set\_yticklabels(["Male", "Female"])  
 # plt.xlabel(["Male", "Female"])  
 # plt.show()  
 ax.set\_ylabel("hands raised")  
  
  
 # == Scatter plots == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 #ax.set\_yticks(np.arange(0, 151, 10))  
 #ax.set\_xticks(np.arange(0, 151, 10))  
 G1, \_ = sp.stats.pearsonr(boyrshndY, boysdiscY)  
 print('Pearsons correlation between raising hands and and participating in discussions amoung boys: %.5f' % G1)  
 G2, \_ = sp.stats.pearsonr(girlrshndY, girlsdsicY)  
 print('Pearsons correlation between raising hands and and participating in discussions amoung girls: %.5f' % G2)  
 ax.scatter(boyrshndY, boysdiscY, alpha=0.2, color="blue")  
 ax.scatter(girlrshndY, girlsdsicY, alpha=0.2, color="pink")  
 # plt.show()  
  
 # == Scatter plots 2 == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 #ax.set\_yticks(np.arange(0, 151, 10))  
 #ax.set\_xticks(np.arange(0, 151, 10))  
 G3, \_ = sp.stats.pearsonr(boyrshndY, boysvisY)  
 print('Pearsons correlation between rasing hands and visiting resources amoung boys: %.5f' % G3)  
 G4, \_ = sp.stats.pearsonr(girlrshndY, girlsvisY)  
 print('Pearsons correlation between rasing hands and visiting resources amoung girls: %.5f' % G4)  
 ax.scatter(boyrshndY, boysvisY, alpha=0.2, color="blue")  
 ax.scatter(girlrshndY, girlsvisY, alpha=0.2, color="pink")  
  
 # == Scatter plot 3 == #  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 #ax.set\_yticks(np.arange(0, 151, 10))  
 #ax.set\_xticks(np.arange(0, 151, 10))  
 G3, \_ = sp.stats.pearsonr(boysdiscY, boysvisY)  
 print('Pearsons correlation between discussion and visiting resources amoung boys: %.5f' % G3)  
 G4, \_ = sp.stats.pearsonr(girlsdsicY, girlsvisY)  
 print('Pearsons correlation between discussion and visiting resources amoung girls: %.5f' % G4)  
 ax.scatter(boysdiscY, boysvisY, alpha=0.2, color="blue")  
 ax.scatter(girlsdsicY, girlsvisY, alpha=0.2, color="pink")  
  
 #plt.show()  
  
 # == for preformance == #  
 # splitting the data  
 ApreformX = data2.loc[data2['SectionID'] == 'A']  
 ApreformY = ApreformX['Discussion'].to\_numpy()  
 ApreformX = ApreformX['SectionID'].to\_numpy()  
  
 BpreformX = data2.loc[data2['SectionID'] == 'B']  
 BpreformY = BpreformX['Discussion'].to\_numpy()  
 BpreformX = BpreformX['SectionID'].to\_numpy()  
  
 CpreformX = data2.loc[data2['SectionID'] == 'C']  
 CpreformY = CpreformX['Discussion'].to\_numpy()  
 CpreformX = CpreformX['SectionID'].to\_numpy()  
  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.bar(ApreformX, ApreformY, label="A Grade")  
 ax.bar(BpreformX, BpreformY, label="B Grade")  
 ax.bar(CpreformX, CpreformY, label="C Grade")  
 ax.legend(loc='best')  
 ax.set\_ylabel("Discussion participation")  
  
 # splitting the data  
 ApreformX = data2.loc[data2['SectionID'] == 'A']  
 ApreformY = ApreformX['AnnouncementsView'].to\_numpy()  
 ApreformX = ApreformX['SectionID'].to\_numpy()  
  
 BpreformX = data2.loc[data2['SectionID'] == 'B']  
 BpreformY = BpreformX['AnnouncementsView'].to\_numpy()  
 BpreformX = BpreformX['SectionID'].to\_numpy()  
  
 CpreformX = data2.loc[data2['SectionID'] == 'C']  
 CpreformY = CpreformX['AnnouncementsView'].to\_numpy()  
 CpreformX = CpreformX['SectionID'].to\_numpy()  
  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.bar(ApreformX, ApreformY, label="A Grade")  
 ax.bar(BpreformX, BpreformY, label="B Grade")  
 ax.bar(CpreformX, CpreformY, label="C Grade")  
 ax.legend(loc='best')  
 ax.set\_ylabel("Announcement views")  
  
 # splitting the data  
 ApreformX = data2.loc[data2['SectionID'] == 'A']  
 ApreformY = ApreformX['raisedhands'].to\_numpy()  
 ApreformX = ApreformX['SectionID'].to\_numpy()  
  
 BpreformX = data2.loc[data2['SectionID'] == 'B']  
 BpreformY = BpreformX['raisedhands'].to\_numpy()  
 BpreformX = BpreformX['SectionID'].to\_numpy()  
  
 CpreformX = data2.loc[data2['SectionID'] == 'C']  
 CpreformY = CpreformX['raisedhands'].to\_numpy()  
 CpreformX = CpreformX['SectionID'].to\_numpy()  
  
 fig = plt.figure()  
 ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])  
 ax.bar(ApreformX, ApreformY, label="A Grade")  
 ax.bar(BpreformX, BpreformY, label="B Grade")  
 ax.bar(CpreformX, CpreformY, label="C Grade")  
 ax.legend(loc='best')  
 ax.set\_ylabel("Hands raised")  
  
  
 #seeing if girls preform better than girls  
 GirlsPreformanceX = data2.loc[data2['Gender'] == 1]  
 GirlsPreformanceY = GirlsPreformanceX['SectionID'].to\_numpy()  
 GirlsPreformanceX = GirlsPreformanceX['Gender'].to\_numpy()  
 BoysPreformanceX = data2.loc[data2['Gender'] == 0]  
 BoysPreformanceY = BoysPreformanceX['SectionID'].to\_numpy()  
 BoysPreformanceX = BoysPreformanceX['Gender'].to\_numpy()  
 newBoysPreform = []  
 for i in BoysPreformanceY:  
 if i == 'A':  
 i = 3  
 if i == 'B':  
 i = 2  
 if i == 'C':  
 i = 1  
 newBoysPreform.append(i)  
 #BoysPreformanceY = [i == 'A' for i in BoysPreformanceY]  
 avgPreformance = sum(newBoysPreform) / len(newBoysPreform)  
 print("The average preformance for Boys: ", avgPreformance)  
  
 newGirlsPreform = []  
 for i in GirlsPreformanceY:  
 if i == 'A':  
 i = 3  
 if i == 'B':  
 i = 2  
 if i == 'C':  
 i = 1  
 newGirlsPreform.append(i)  
 #BoysPreformanceY = [i == 'A' for i in BoysPreformanceY]  
 avgGirlsPreformance = sum(newGirlsPreform) / len(newGirlsPreform)  
 print("The average preformance for Girls: ", avgGirlsPreformance)  
  
 # ax.bar(BpreformX, BpreformY, label="")  
 plt.show()  
 # data2.to\_csv("Test.csv")

Question 2:

import numpy as np  
import pandas as pd  
from scipy.spatial.distance import hamming, jaccard, euclidean  
from scipy.sparse.csgraph import minimum\_spanning\_tree  
from relativeNeighborhoodGraph import returnRNG as cRNG  
from sigfig import round  
  
  
def \_init\_():  
 print("starting E2")  
 rawNames = pd.read\_excel(io='Datasets/AlzheimersDisease.xls', sheet\_name="Training Set", usecols="A")  
 NProtienes = rawNames.values.flatten()  
  
 rawNames = pd.read\_excel(io='Datasets/AlzheimersDisease.xls', sheet\_name="Training Set")  
 NSamples = rawNames.columns.values.tolist()  
 NSamples = NSamples[1:]  
 print(len(NSamples))  
  
 rawSamples = pd.read\_excel(io='Datasets/AlzheimersDisease.xls', sheet\_name="Training Set", usecols="B:CF").to\_numpy()  
  
 rawProteins = pd.read\_excel(io='Datasets/AlzheimersDisease.xls', sheet\_name="Training Set", usecols="B:CF").to\_numpy()  
  
 rawSamples = np.transpose(rawSamples)  
 # print(rawSamples)  
 HammSamplesEmpty = np.zeros([rawSamples.shape[0], rawSamples.shape[0]], dtype=float)  
 HammProteinsEmpty = np.zeros([rawProteins.shape[0], rawProteins.shape[0]], dtype=float)  
  
 hammingSamples = calcMatrix(data=rawSamples, matrix=HammSamplesEmpty, offset=0, type=0)  
 hammingProteins = calcMatrix(data=rawProteins, matrix=HammProteinsEmpty, offset=0, type=0)  
  
 genMST(matrix=hammingSamples, index=NSamples, exportLoc="./Answers/E2SamplesMST.xlsx")  
 genMST(matrix=hammingProteins, index=NProtienes, exportLoc="./Answers/E2ProteinsMST.xlsx")  
  
 genRNG(data=hammingSamples, index=NSamples, exportLoc="./Answers/E2SamplesRNG.xlsx")  
 genRNG(data=hammingProteins, index=NProtienes, exportLoc="./Answers/E2ProteinsRNG.xlsx")  
  
 # print(rawProteins)  
 # print(rawSamples)  
  
 # just gunna use a hemming matrix  
  
  
def genMST(matrix, index, exportLoc):  
 # This is magic to me xD  
 Tcsr = minimum\_spanning\_tree(matrix)  
 # print(matrix)  
 Tcsr = Tcsr.toarray()  
 final = pd.DataFrame(data=Tcsr, columns=index, index=index, dtype=float)  
 # print(Tcsr)  
 final.to\_excel(exportLoc)  
 return Tcsr  
  
  
def calcMatrix(data, matrix, offset, type):  
 i = 0  
 # print(data.shape)  
 while i < data.shape[0]:  
 x = i  
 # base = makeUseable(np.array2string(data[i]))  
 # print("str test: " + base)  
 while x < data.shape[0]:  
 if type == 0:  
 temp = euclidean(data[i], data[x])  
 #temp = round(temp, sigfig=5)  
 # print(str(i) + " " + str(x) + " " + str(temp) + str(data[i]) + " " + str(data[x]))  
 else:  
 temp = hamming(data[i], data[x]) \* len(data[i])  
 matrix[i + offset, x + offset] = temp  
 matrix[x + offset, i + offset] = temp  
 # print(hamming(data[i], data[x]) \* len(data[i]))  
 x += 1  
 i += 1  
 return matrix  
  
  
def genRNG(data, index, exportLoc):  
 # print(data)  
 rng = cRNG.returnRNG(data)  
 # print(rng)  
 rng = rng.to\_numpy()  
 rng = remDoubleLines(rng)  
 final = pd.DataFrame(data=rng, columns=index, index=index, dtype=float)  
 # print(Tcsr)  
 final.to\_excel(exportLoc)  
 return  
  
  
def remDoubleLines(matrix):  
 X = 0  
 Y = 0  
 while X < matrix.shape[1]:  
 Y = 0  
 while Y < matrix.shape[0]:  
 if matrix[X, Y] != 0:  
 if matrix[X, Y] == matrix[Y, X]:  
 # print(matrix[X, Y])  
 matrix[X, Y] = 0  
 Y += 1  
 X += 1  
 # print(str(X) + " " + str(Y))  
 return matrix

Quesiton3:

from sklearn.feature\_selection import SelectKBest  
import sklearn.preprocessing as pp  
from sklearn.feature\_selection import chi2  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import metrics  
import numpy as np  
import pandas as pd  
  
  
def Begin():  
 # === Feture selection === #  
 # Gathering data  
 rawData = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Training Set", usecols="B:CF").to\_numpy()  
 protienes = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Training Set", usecols="A")  
 rawNames = pd.read\_excel(io='Datasets/AlzheimersDisease.xls', sheet\_name="Training Set")  
 NSamples = rawNames.columns.str.split(".").str[0]  
 NSamples = NSamples[1:]  
 # print(NSamples)  
 # normalising all the data usking sklearn's preprocessing  
 # normData = pp.normalize(rawData)  
 Scalar = pp.MinMaxScaler()  
 #print(rawData.shape)  
 normData = np.transpose(rawData)  
 # print(normData.shape)  
 normData = Scalar.fit\_transform(normData)  
 label\_encode = pp.LabelEncoder()  
  
 X = normData[:, 0: 121]  
 Y = normData[:, -1]  
 #print(Y)  
 Y = label\_encode.fit\_transform(Y)  
 # Extracting the best featurs  
 bestFeatures = SelectKBest(score\_func=chi2, k=120)  
 fit = bestFeatures.fit\_transform(X, Y) # These are the final selected features.  
 # print(fit.shape)  
 # preping it for use  
 dfScores = pd.DataFrame(fit)  
 dflabel = pd.DataFrame(NSamples)  
 finalFeaturs = pd.concat([dflabel, dfScores], axis=1)  
 # finalFeaturs.columns = ['Illness', 'Score']  
 finalFeaturs.to\_csv('Test.csv')  
 # print(finalFeaturs)  
 # finalFeatures = fit.scores\_  
  
 # === Classifying the data === #  
 model = KNeighborsClassifier(n\_neighbors=3)  
 # -- training the model -- #  
 model.fit(fit, NSamples) # it has learned lots  
  
 #testing it.  
 # X\_train, X\_test, Y\_train, Y\_test = train\_test\_split()  
  
 # == Testing it with the other Datasets == #  
 # importing data to test it with. i am removeing the CD class as i have not trained the classifier to deal with them.  
 rawTestData = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Test Set AD", usecols="B:CD").to\_numpy()  
 testData = rawTestData.transpose()  
 X\_Test = testData[:, 0: 120]  
 #X\_Test = Scalar.fit\_transform(X\_Test)  
 Y\_Test = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Test Set AD", usecols="B:CE")  
 Y\_Test = Y\_Test.columns.str.split(".").str[0]  
 # print(Y\_Test)  
 Y\_Test = Y\_Test[1:]  
 #testing the model  
 # Y\_pred60 = model.predict(X\_Test[:, 0: 60])  
 Y\_pred120 = model.predict(X\_Test[:, 0:120])  
 # print (X\_Test[:, 0: 60])  
 #print (Y\_Test.shape)  
 # graph = model.kneighbors\_graph()  
 print("== Results for the test set ==")  
 #print("Accuracy 0-60: ", metrics.accuracy\_score(Y\_Test, Y\_pred60))  
 print("Accuracy: ", metrics.accuracy\_score(Y\_Test, Y\_pred120))  
 #print("Mathews Correlation 0-60: ", metrics.matthews\_corrcoef(Y\_Test, Y\_pred60))  
 print("Mathews Correlation: ", metrics.matthews\_corrcoef(Y\_Test, Y\_pred120))  
 # print("F-1 Score 0-60: ", metrics.f1\_score(Y\_Test, Y\_pred60, average='weighted'))  
 print("F-1 Score: ", metrics.f1\_score(Y\_Test, Y\_pred120, average='weighted'))  
 # for spesificity & sensativity.  
 TN, FP, FN, TP = metrics.confusion\_matrix(Y\_Test, Y\_pred120, labels=["AD","NDC"]).ravel()  
 spc =TN/(TN+FP)  
 sen =TP/(TP + FN)  
 print("Specificity: ", spc)  
 print("Sensitivity: ", sen)  
 # print(rawProteins)  
  
 # ==== going for MCI samples === #  
 rawTestData = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Test Set MCI", usecols="B:W", skiprows=[0]).to\_numpy()  
 testData = rawTestData.transpose()  
  
 Y\_Test = pd.read\_excel(io="Datasets/AlzheimersDisease.xls", sheet\_name="Test Set AD", usecols="B:CE")  
 Y\_Test = Y\_Test.columns.str.split(".").str[0]  
 Y\_Test = Y\_Test[:22]  
 #print(Y\_Test)  
  
 TestData = testData[:, 0: 120]  
 # TestData = np.delete(TestData, 0, 1)  
  
 Y\_pred120 = model.predict(TestData)  
 # TestData = pd.DataFrame(TestData)  
 # TestData.to\_csv("Test.csv")  
 # print(rawTestData)  
 print("\n== Results for the test set ==")  
 #print("Accuracy 0-60: ", metrics.accuracy\_score(Y\_Test, Y\_pred60))  
 print("Accuracy: ", metrics.accuracy\_score(Y\_Test.to\_numpy(), Y\_pred120))  
 #print("Mathews Correlation 0-60: ", metrics.matthews\_corrcoef(Y\_Test, Y\_pred60))  
 print("Mathews Correlation: ", metrics.matthews\_corrcoef(Y\_Test.to\_numpy(), Y\_pred120))  
 # print("F-1 Score 0-60: ", metrics.f1\_score(Y\_Test, Y\_pred60, average='weighted'))  
 print("F-1 Score: ", metrics.f1\_score(Y\_Test.to\_numpy(), Y\_pred120, average='weighted'))  
 # for spesificity & sensativity.  
 TN, FP, FN, TP = metrics.confusion\_matrix(Y\_Test.to\_numpy(), Y\_pred120, labels=["AD","NDC"]).ravel()  
 spc =TN/(TN+FP)  
 sen =TP/(TP + FN)  
 print("Specificity: ", spc)  
 print("Sensitivity: ", sen)  
 # print(rawProteins)  
 # print(Y\_Test)  
 return