**Task one: Distance and similarity calculation (10 points)**

Formula Used:cos(dy d2) = (d1\*d2) / ( |d1| \* |d2| )

d1 \* d2 = (0\*5) + (4\*19) + (10\*7) + (8\*16) + (0\*0) + (5\*0) + (0\*32) = 274

|d1| = [ (0\*0) + (4\*4) + (10\*10) + (8\*8) + (0\*0) + (5\*5) + (0\*0) ].5 = 205.5

|d2|= [ (5\*5) + (19\*19) + (7\*7) + (16\*16) + (0\*0) + (0\*0) + (32\*32) ].5 = 1715.5

cos(dy d2) = 274 / (205.5 \*1715.5) = .46210596

**Final Answer: .46210596**

**Task two: Probability and Bayes rule (20 points)**



Formula : =

=

**Math for P(A):**

( ( (.95) \* (.01) ) / (.1085) ) / (.01) = (.08755) / (.01) =***answer = 8.7557***

**Task three: PCA analysis simulation (25 points)**

CODE:

SETUP (Code provided below)

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.decomposition import PCA  
from numpy.linalg import eig  
  
#--------------------------- Description --------------------------#  
# #  
# This program creates three graphs, 1) a randomly distributed #  
# scatter plot with a regression line, 2) 1st PCA transform, #  
# and 3) reconstruction to 2d scatter plot. #  
# #  
#------------------------------------------------------------------#  
  
#---------------------------- SETUP -------------------------------#  
rng = np.random.RandomState(298) # creates a seed for the random numbers, to keep results consistent  
X = np.dot(rng.rand(2, 2), rng.randn(2, 1000)).T # creates 1000 points with normal distribution  
# Find the slope and intercept of the best fit line  
slope, intercept = np.polyfit(X[:, 0], X[:, 1], 1) # regression on matrix X[x], X[y], sets slope, intercept  
pca = PCA(n\_components=1) # sets dimension to 1st  
pca.fit(X)  
X\_pca = pca.transform(X)  
X\_new = pca.inverse\_transform(X\_pca)

[2 points] # generate 1000 randomly distributed data points with x and y two features/dimensions (Code provided below)

#---------------------------- PART ONE ----------------------------#  
# Generate fake data -- 1000 data points generated in a random fashion  
plt.figure(1)  
plt.title("Original Data")  
plt.plot(X[:, 0], X[:, 0]\*slope + intercept, 'r') # plots line  
plt.scatter(X[:, 0], X[:, 1], c='b', s=8, alpha= 1) # creates a scatter plot with 1000 points

[10 points] # calculate the first principle component and project the data points onto it (Code provided below)

#---------------------------- PART TWO ----------------------------#  
# DEFINE THE ORIGINAL MATRIX  
O = X\_new # O = original matrix  
# CALCULATE MEAN OF MATRIX  
M = np.mean(O.T, axis=1) # M = mean matrix  
# CENTER MATRIX AROUND 0,0  
C = O - M # C = centered matrix  
# CALCULATE COVARIANCE MATRIX  
V = np.cov(C.T) # V = covariance matrix  
# EIGEN COMPOSITION OF COVARIANCE MATRIX  
values, vectors = eig(V) # eigen values and vectors  
# FINAL FIRST PCA DATA  
P = vectors.T.dot(C.T) # P = first pca of data  
# PLOTTING  
plt.figure(2)  
plt.title("First PC")  
plt.scatter(P.T[:, 0], P.T[:, 1], c='r', s=8) # plot P[x], P[y], color red, size 8

[3 points] # reconstruct the original points from this one principal component

#---------------------------- PART THREE ----------------------------#  
# PLOTTING  
plt.figure(3)  
plt.title("Reconstruction")  
plt.scatter(X\_new[:, 0], X\_new[:, 1], c='k', s=8) # plot matrix X[x], X[y], color black, size 8  
plt.show()

**Report on Project**

**Steps:**

For this project, I broke this part of the homework into 3 separate steps which would allow me to focus on figuring out how to replicate the graphs.

*Step 1*

To create the first plot, I researched ways to generate data on a line and discovered many unique ways. I finally settled on a using a random seed generator (set to 298 in the code as it seemed like a pretty good distribution) to be able to test my code. This would allow me to create a distribution around a line that could also be placed into a matrix for future use. From there, I used the numpy’s polyfit function to find the slope and intercept of the distribution of the scatter plot. Finally, plotting both on figure one allowed me to see the line and scatter plot as required.

*Step 2*

The second plot was a little more difficult to produce, I actually worked on transforming the data into the third plot and then using eigen values to convert the reconstruction plot back to the First PC. To do this, I set the dimension of the PCA to 1 and then have PCA fit to the original matrix. From there, I transformed the pca matrix and then inversed it to get the Reconstruction graph. For the First PC plot, I used the algorithm from the slides for dimension reduction with the help of online resources to help use matrixes correctly. After modifying the matrixes, I was able to replicate the First PC provided by the professor for the dataset that I had created.

*Step 3*

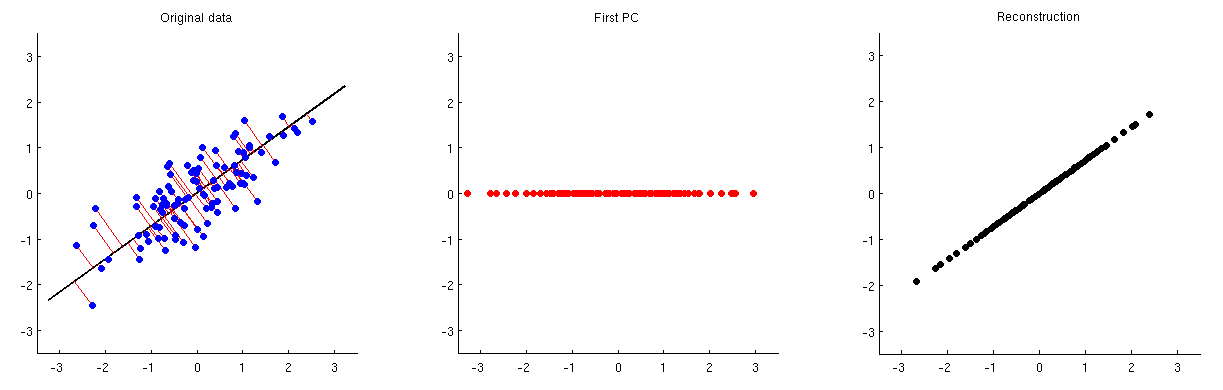
To complete the reconstruction, I set the dimension of the PCA to 1 and then have PCA fit to the original matrix. From there, I transformed the pca matrix and then inversed it to get the Reconstruction graph. With the graph constructed, I just plotted it with results that looked very similar to the professor’s.

**Results:**

The results of the project are generated in three different plots, figure1, figure 2, and figure 3. These three figures show how the distribution looks like with a regression line, dimension reduction, and final reconstruction. Overall, the results looked incredibly similar to the provided example for which I feel I have successfully replicated the general steps that the professor used to create his plots.

**Plots:**

Provided Graphs by the Teacher below:



This project’s goal was to replicate the graphs above. The code above will provide the user with very similar graphs that are only different due to slight changes in slope of the regression line. Below is the plot that shows the original data with 1000 data points and a regression line. There are some slight differences between this scatter plot than the provided including the regression line which in the original goes past the data points while in my example, only stretches to the farthest data points.



For the second graph, one can see more similarities as most points fall on a normal distribution along the x axis.



Finally, the last plot shows the transformation of the data back to 2d with all points on the regression line. This plot looks incredibly similar to the provided example.



**Your Thoughts on this Task**

Originally, I thought this task would be quite difficult as my concept of creating graphs and charts were quite complicated using Java. However, by using Python and its class libraries, this task was not very complicated (but still took some time to complete). Overall, I really enjoyed working on this project as it helped me get more used to the class libraries that are needed for machine learning and how to create plots. It also helped me learn how to look for helpful material to complete the project as I was not familiar with all the methods within the class libraries. Ultimately, I think this was a great first programming project that helped me understand how to do linear regression on a randomly generated distribution and how to use matrixes to reduce dimensions.

**Coding References:**

- <https://stackoverflow.com/questions/7941226/how-to-add-line-based-on-slope-and-intercept-in-matplotlib>

\* for regression line and slope/intercept

- <https://stackoverflow.com/questions/51064456/principal-component-analysis-dimension-reduction-in-python>

\* for Part Two help with matrix transformation

- <https://canvas.uw.edu/courses/1212093/files/folder/Slides?preview=50207056>

\* for Part Two help with matrix transformation algorithm

- <https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html>

\* for help with PCA transformation (part 3)