Yu Long Wang
Homework 1 report

1. For problem 1 the loop version we loop through the length of rows and cols.
   Using two for loops gets the entire row (1 x N) and performs the summation of the difference on two rows on it to the power of two and squares root it for the final distance. We explicitly calculate the distance for each [row, col] using euclidean distance (sqrt(xi - xj)) and assign it. Eventually it becomes an N x N matrix.

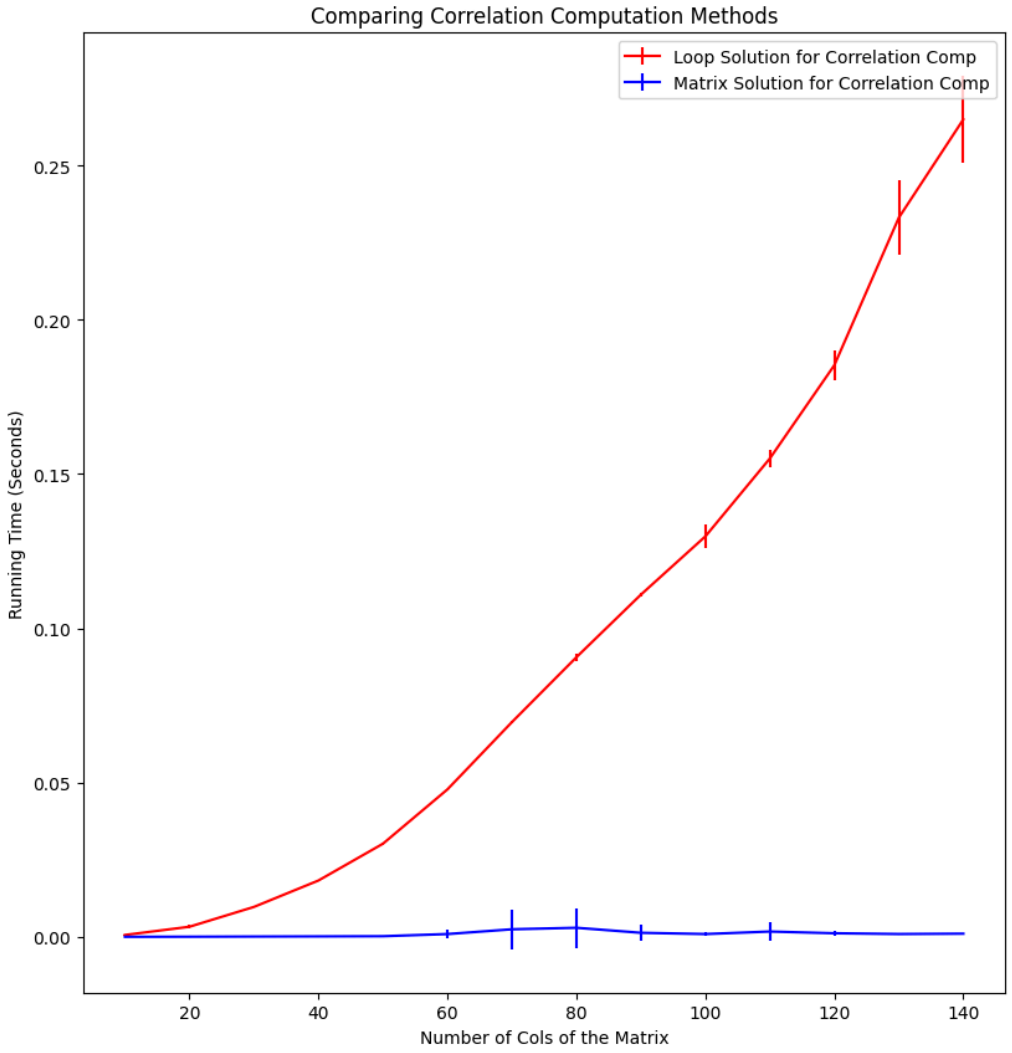
   The vectorization version doesn't use loops. It calculates the squared norms all at once for each row in the matrix and becomes a 1 x D matrix. We prepare to calculate the distance by broadcasting the squared norms which allows us to virtually change shapes of vectors without loops changing the row into 1 x N and column to N x 1, which is also later expanded to N x N when it computes with the dot product of X and X transpose which is already N x N. The dot product X and X transpone calculates the product for all pairwise elements at the same time instead of using a loop. Once done we square root the summation of the row subtracted by two times the dot product and add the column. This is also all broadcasted and performs at the same time with loops.

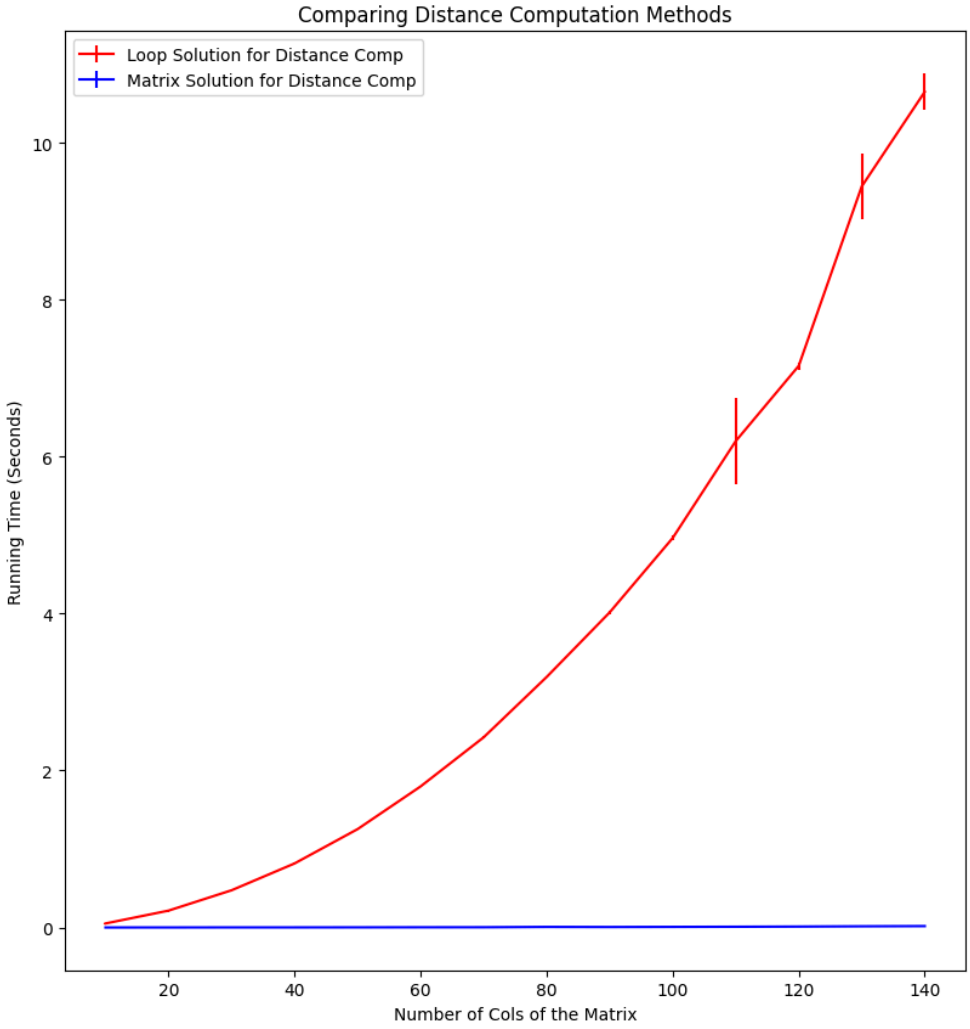
   For the native correlation we use loops to calculate the mean, covariance, standard deviation, and correlation. When calculating the mean it iterates over all the columns and gets the summation for the jth column and is divided by N, storing into an array of means.
   The covariance uses two loops to calculate each covariance by dividing N - 1 by the summation of (xi - mu[i]) - [xj - mu[j]) by their respective ith and jth column and mean. We each loop storing the result into M[i][j]. Sigma also uses a loop to iterate over and get the square root for the diagonal. The correlation uses two loops for i and j, to calculate the covariance divided by its respective sigma of i and j (sigma[i] * sigma[j]) and storing for each loop into the matrix one at a time.

   The vectorization method removes the loops and performs the calculation all at the same time. We get the mean with np.mean(x, 0) to get the mean of the column, and mean() calculates all the columns at the same time in a single operation and in compiled code which is faster. We then calculate the covariance by getting the dot product of X - mean transposed (D x N) and X - mean (N x D), which is also broadcasted so that mean can be subtracted from each row, where mean is 1 x D. Which results in a size of D x D and each is divided by N - 1 at the same time. The standard deviation is also vectorized by getting the square root of the diagonal of covariance without a loop. We then get the pairwise standard deviations since [i,j] = sigma[i] * sigma[j], and to do this we need to expand the dimension to convert the array into a vector, where row vector is 1 X D and column

vector is D x 1. This allows us to get the dot product which is pairwise (D x D). We would then divide the pairwise with the covariance to get the correlation for each point all at the same time without a loop.

2. Whenever we use loops we can see that as the number of columns increases the computation time required with loops seems to grow exponentially. In comparison with vectorization, even with the increase of columns the amount of time that is required to perform the computations is significantly slower and more consistent. The control variable in this experiment is the number of columns that we have. Overall the amount of time that is required to perform calculations with loops increases as we have a larger dataset. Vectorization on the other hand still remains quick despite the increase of data as we can see with the digits computations, 15 compared to .02.






```
iris data shape:  (150, 4)
breast cancer data shape:  (569, 30)
digits data shape:  (1797, 64)
Dataset              Loops    Vectorization
-----------------------------------------
iris              0.106352         0.000381
breast_cancer     1.548143         0.004628
digits           15.793917         0.029398
```