```
function [U, S] = pca(X)
%PCA Run principal component analysis on the dataset X
%   [U, S, X] = pca(X) computes eigenvectors of the covariance matrix of X
%   Returns the eigenvectors U, the eigenvalues (on diagonal) in S
%

% Useful values
[m, n] = size(X);

% You need to return the following variables correctly.
U = zeros(n);
S = zeros(n);

% ====================== YOUR CODE HERE ======================
% Instructions: You should first compute the covariance matrix. Then, you
%               should use the "svd" function to compute the eigenvectors
%               and eigenvalues of the covariance matrix.
%
% Note: When computing the covariance matrix, remember to divide by m (the
%       number of examples).
%
% DIMENSIONS:
%    X = m x n
```

```
Sigma = (1/m)*(X'*X);
[U, S, V] = svd(Sigma);






% =========================================================================

end
```

```
function Z = projectData(X, U, K)
%PROJECTDATA Computes the reduced data representation when projecting only
%on to the top k eigenvectors
%   Z = projectData(X, U, K) computes the projection of
%   the normalized inputs X into the reduced dimensional space spanned by
%   the first K columns of U. It returns the projected examples in Z.
%

% You need to return the following variables correctly.
Z = zeros(size(X, 1), K);

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the projection of the data using only the top K
%               eigenvectors in U (first K columns).
%               For the i-th example X(i,:), the projection on to the k-th
%               eigenvector is given as follows:
%                    x = X(i, :)';
%                    projection_k = x' * U(:, k);
%
U_reduce = U(:, 1:K);
Z = X * U_reduce;
```

```
function X_rec = recoverData(Z, U, K)
%RECOVERDATA Recovers an approximation of the original data when using the
%projected data
%   X_rec = RECOVERDATA(Z, U, K) recovers an approximation the
%   original data that has been reduced to K dimensions. It returns the
%   approximate reconstruction in X_rec.
%

% You need to return the following variables correctly.
X_rec = zeros(size(Z, 1), size(U, 1));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the approximation of the data by projecting back
%               onto the original space using the top K eigenvectors in U.
%
%               For the i-th example Z(i,:), the (approximate)
%               recovered data for dimension j is given as follows:
%                    v = Z(i, :)';
%                    recovered_j = v' * U(j, 1:K)';
%
%               Notice that U(j, 1:K) is a row vector.
%
```

```
%
U_reduce = U(:, 1:K);
X_rec = Z * U_reduce';


% =============================================================

end
```

```
function idx = findClosestCentroids(X, centroids)
%FINDCLOSESTCENTROIDS computes the centroid memberships for every example
%   idx = FINDCLOSESTCENTROIDS (X, centroids) returns the closest centroids
%   in idx for a dataset X where each row is a single example. idx = m x 1
%   vector of centroid assignments (i.e. each entry in range [1..K])
%

% Set K
K = size(centroids, 1);

% You need to return the following variables correctly.
idx = zeros(size(X,1), 1);

% ====================== YOUR CODE HERE ======================
% Instructions: Go over every example, find its closest centroid, and store
%               the index inside idx at the appropriate location.
%               Concretely, idx(i) should contain the index of the centroid
%               closest to example i. Hence, it should be a value in the
%               range 1..K
%
% Note: You can use a for-loop over the examples to compute this.
```

```
%
for i =1:length(X)
    x = X(i,:);
    norms = zeros(K, 1);
    for centroid_i = 1: K
        norms(centroid_i) = (x - centroids(centroid_i,:)) * (x - centroids(centroid_i, :))';
    end
    [value, idx(i)] = min(norms);
end
```

```
function centroids = computeCentroids(X, idx, K)
%COMPUTECENTROIDS returns the new centroids by computing the means of the
%data points assigned to each centroid.
%   centroids = COMPUTECENTROIDS(X, idx, K) returns the new centroids by
%   computing the means of the data points assigned to each centroid. It is
%   given a dataset X where each row is a single data point, a vector
%   idx of centroid assignments (i.e. each entry in range [1..K]) for each
%   example, and K, the number of centroids. You should return a matrix
%   centroids, where each row of centroids is the mean of the data points
%   assigned to it.
%

% Useful variables
[m n] = size(X);

% You need to return the following variables correctly.
centroids = zeros(K, n);


% ====================== YOUR CODE HERE ======================
% Instructions: Go over every centroid and compute mean of all points that
%               belong to it. Concretely, the row vector centroids(i, :)
%               should contain the mean of the data points assigned to

%               centroid i.
%
% Note: You can use a for-loop over the centroids to compute this.
%
for i=1 :K
    centroids(i, :) = mean(X([find(idx == i)], :));
end
```

```
function centroids = kMeansInitCentroids(X, K)
%KMEANSINITCENTROIDS This function initializes K centroids that are to be
%used in K-Means on the dataset X
%   centroids = KMEANSINITCENTROIDS(X, K) returns K initial centroids to be
%   used with the K-Means on the dataset X
%

% You should return this values correctly
centroids = zeros(K, size(X, 2));

% ====================== YOUR CODE HERE ======================
% Instructions: You should set centroids to randomly chosen examples from
%               the dataset X
%


% Randomly reorder the indices of examples
randidx = randperm(size(X, 1));

% Take the first K example as centroids
centroids = X(randidx(1:K), :);
```