Table of Contents

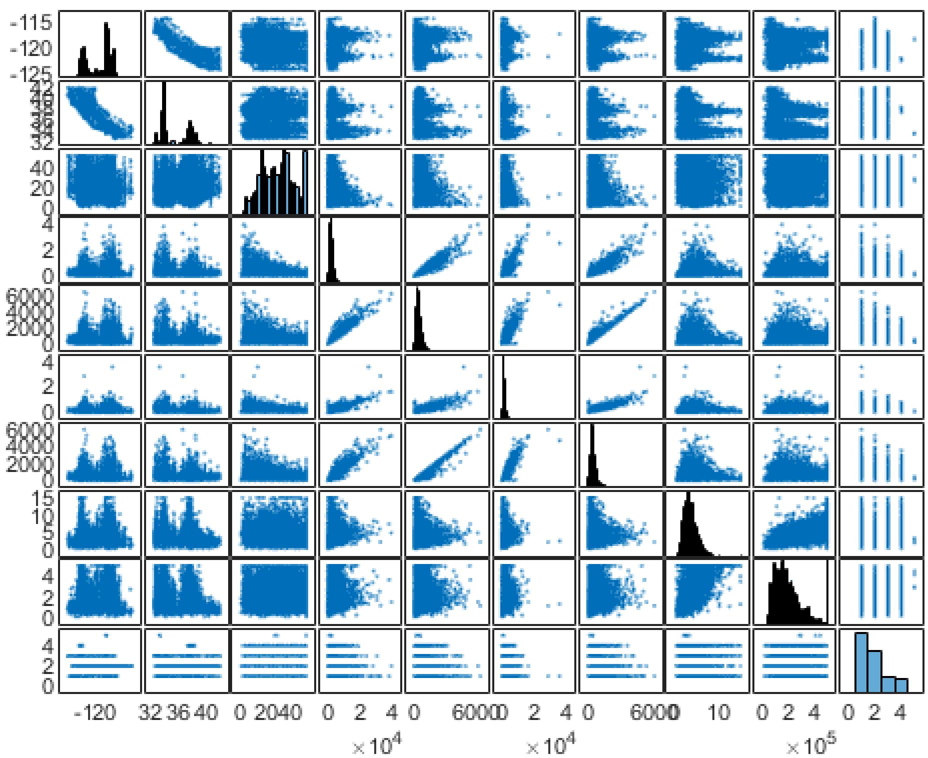
BE606 HW3 Problem 1

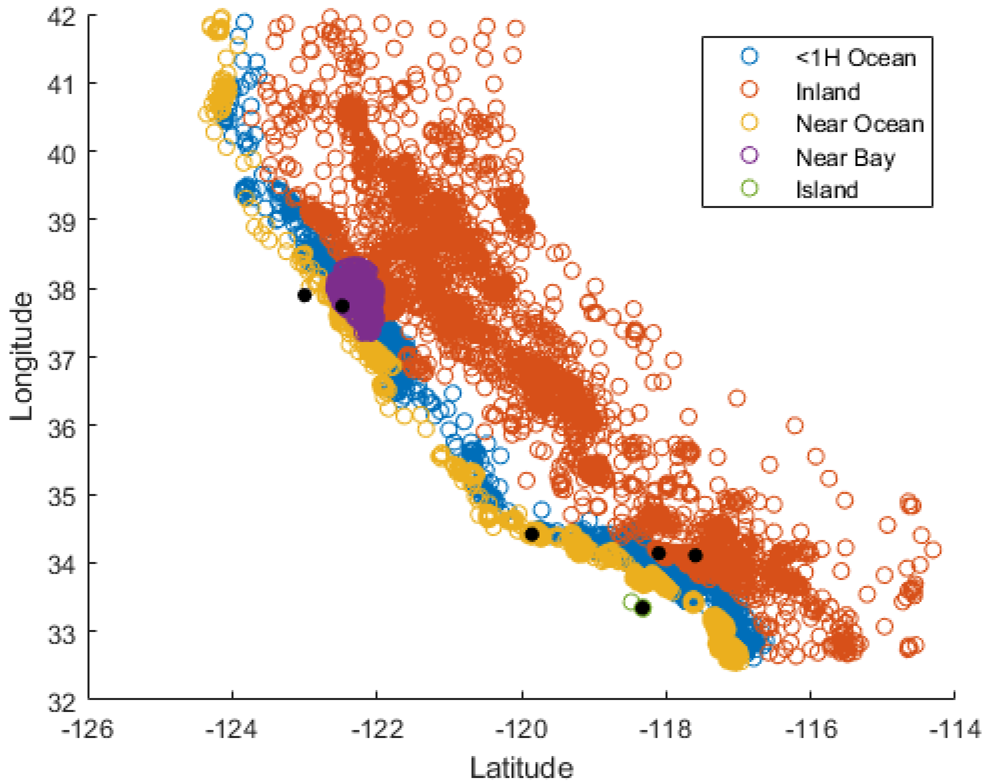
```
close all
clear all
```

Part 1

```
A = readtable('housing.csv');


for i = 1:1:20640
    if strcmp(A.ocean_proximity(i), 'NEAR BAY')
        A.ocean_proximity(i) = strrep(A.ocean_proximity(i), 'NEAR
 BAY', '4');
    elseif strcmp(A.ocean_proximity(i), '<1H OCEAN')
        A.ocean_proximity(i) = strrep(A.ocean_proximity(i), '<1H
 OCEAN', '1');
    elseif strcmp(A.ocean_proximity(i), 'INLAND')
        A.ocean_proximity(i) =
 strrep(A.ocean_proximity(i), 'INLAND', '2');
    elseif strcmp(A.ocean_proximity(i), 'NEAR OCEAN')
        A.ocean_proximity(i) = strrep(A.ocean_proximity(i), 'NEAR
 OCEAN', '3');
    else
        A.ocean_proximity(i) =
 strrep(A.ocean_proximity(i), 'ISLAND', '5');
    end
%     str2double(A.ocean_proximity(i));
end


OPClass = A.ocean_proximity;
abc = cellfun(@str2num,OPClass);
B = table2array(A(:,1:9));

B = [B abc];

figure;
plotmatrix(B)


x1 = B(:,1);
```

```
x2 = B(:,2);
y = B(:,10);
figure;


for kk = 1:5
    scatter(x1(y == kk), x2(y == kk))
    hold on
end


legend('<1H Ocean', 'Inland', 'Near Ocean', 'Near Bay', 'Island')
xlabel('Latitude')
ylabel('Longitude')


%new data

x1new = [-117.59292, -122.99700, -122.47476, -118.10267, -119.85405,
 -118.32575];
x2new = [34.10626, 37.89909, 37.74269, 34.13808, 34.41536, 33.34261];


scatter(x1new,x2new,30, 'ko', 'filled','HandleVisibility','off')

hold off
```





knnsearch

```
X = [x1 x2];
Ynew = [x1new' x2new'];

[idx, eD] = knnsearch(X,Ynew,'K', 20);

house1idx = idx(1,:)';
house1x1 = x1(house1idx);
house1x2 = x2(house1idx);
house1y = y(house1idx);

house2idx = idx(2,:)';
house2x1 = x1(house2idx);
house2x2 = x2(house2idx);
house2y = y(house2idx);

house3idx = idx(3,:)';
house3x1 = x1(house3idx);
house3x2 = x2(house3idx);
house3y = y(house3idx);

house4idx = idx(4,:)';
house4x1 = x1(house4idx);
house4x2 = x2(house4idx);
```

```
house4y = y(house4idx);

house5idx = idx(5,:)';
house5x1 = x1(house5idx);
house5x2 = x2(house5idx);
house5y = y(house5idx);

house6idx = idx(6,:)';
house6x1 = x1(house6idx);
house6x2 = x2(house6idx);
house6y = y(house6idx);

houseclasstot = [mode(house1y) mode(house2y) mode(house3y)
 mode(house4y) mode(house5y) mode(house6y)];
%make into table for output
for jj = 1:6
    fprintf('New House #%d ',jj)
    fprintf('Classified as %d\n', houseclasstot(jj))
    disp('')
end


House1table=table(house1x1, house1x2, house1y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
House2table=table(house2x1, house2x2, house2y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
House3table=table(house3x1, house3x2, house3y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
House4table=table(house4x1, house4x2, house4y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
House5table=table(house5x1, house5x2, house5y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
House6table=table(house6x1, house6x2, house6y, 'VariableNames',
 {'Longitude','Latitude', 'HousingClass'})
```

```
New House #1 Classified as 2
New House #2 Classified as 3
New House #3 Classified as 4
New House #4 Classified as 1
New House #5 Classified as 3
New House #6 Classified as 3

House1table =

  20×3 table

    Longitude    Latitude    HousingClass
    _________    ________    ____________

     -117.59       34.1           2
      -117.6      34.11           2
     -117.58      34.11           2
     -117.58       34.1           2
     -117.59      34.09           2
```

-117.61	34.1	2
-117.58	34.09	2
-117.61	34.12	2
-117.61	34.09	2
-117.61	34.09	2
-117.59	34.13	2
-117.6	34.08	2
-117.62	34.11	2
-117.62	34.11	2
-117.61	34.13	2
-117.61	34.08	2
-117.61	34.08	2
-117.62	34.09	2
-117.57	34.13	2
-117.56	34.12	2

House2table =

20×3 table

Longitude	*Latitude*	*HousingClass*
-122.93	38.02	3
-122.84	38.07	3
-122.86	38.1	3
-122.81	38.08	3
-122.71	37.9	3
-122.71	37.88	3
-122.69	37.91	3
-122.7	38.03	3
-122.68	38.01	3
-122.66	37.93	3
-122.8	38.18	3
-122.68	38.07	3
-122.64	37.96	3
-122.96	38.26	3
-122.65	38.01	3
-122.64	38.01	3
-122.64	38.01	3
-122.62	37.85	3
-122.62	37.97	3
-122.9	38.28	3

House3table =

20×3 table

Longitude	*Latitude*	*HousingClass*
-122.47	37.74	4

-122.47	37.74	4
-122.47	37.74	4
-122.47	37.74	4
-122.48	37.74	3
-122.48	37.74	3
-122.48	37.74	3
-122.48	37.74	3
-122.47	37.75	4
-122.47	37.75	4
-122.47	37.75	4
-122.47	37.75	4
-122.47	37.75	4
-122.47	37.75	4
-122.48	37.75	4
-122.48	37.75	4
-122.48	37.75	4
-122.48	37.75	4
-122.48	37.75	4
-122.47	37.73	3

House4table =

20×3 table

Longitude	Latitude	HousingClass
-118.1	34.14	1
-118.1	34.14	1
-118.11	34.14	1
-118.11	34.14	1
-118.1	34.13	1
-118.1	34.13	1
-118.1	34.13	1
-118.1	34.15	2
-118.1	34.15	2
-118.09	34.14	2
-118.11	34.15	1
-118.11	34.15	1
-118.09	34.15	2
-118.09	34.15	2
-118.09	34.15	2
-118.09	34.15	2
-118.12	34.14	1
-118.12	34.14	1
-118.1	34.12	1
-118.1	34.12	1

House5table =

20×3 table

Longitude	Latitude	HousingClass
-119.86	34.42	3
-119.86	34.41	3
-119.85	34.4	3
-119.85	34.44	3
-119.88	34.42	3
-119.83	34.43	3
-119.84	34.44	3
-119.88	34.43	3
-119.88	34.43	3
-119.88	34.43	3
-119.88	34.4	3
-119.83	34.44	3
-119.83	34.44	3
-119.88	34.44	3
-119.86	34.38	3
-119.86	34.38	3
-119.82	34.43	3
-119.84	34.45	3
-119.82	34.44	3
-119.82	34.44	3

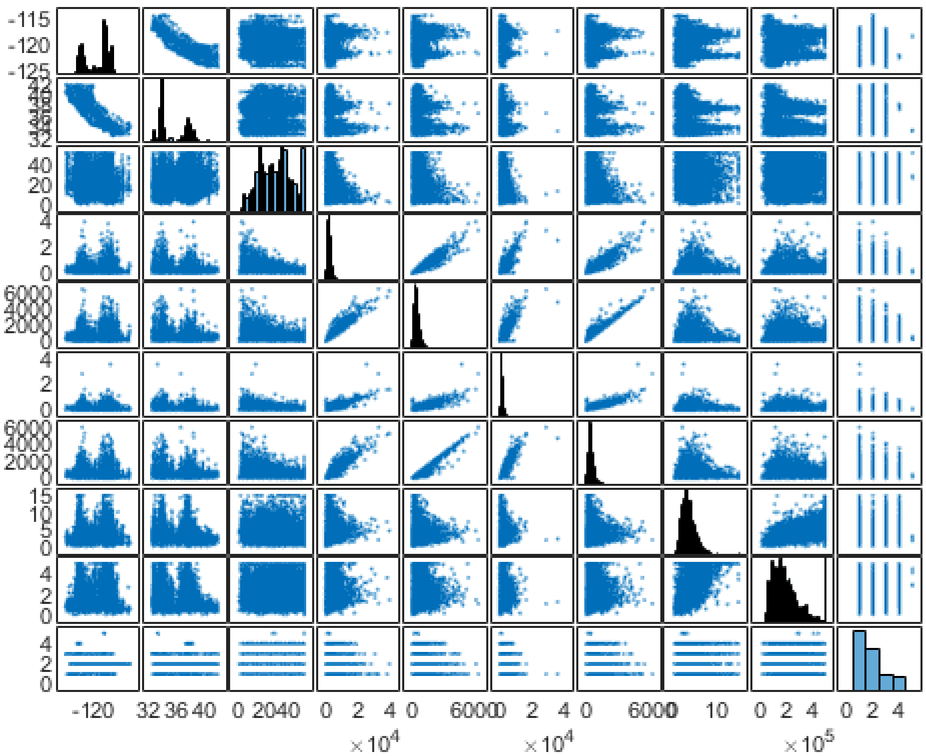
House6table =

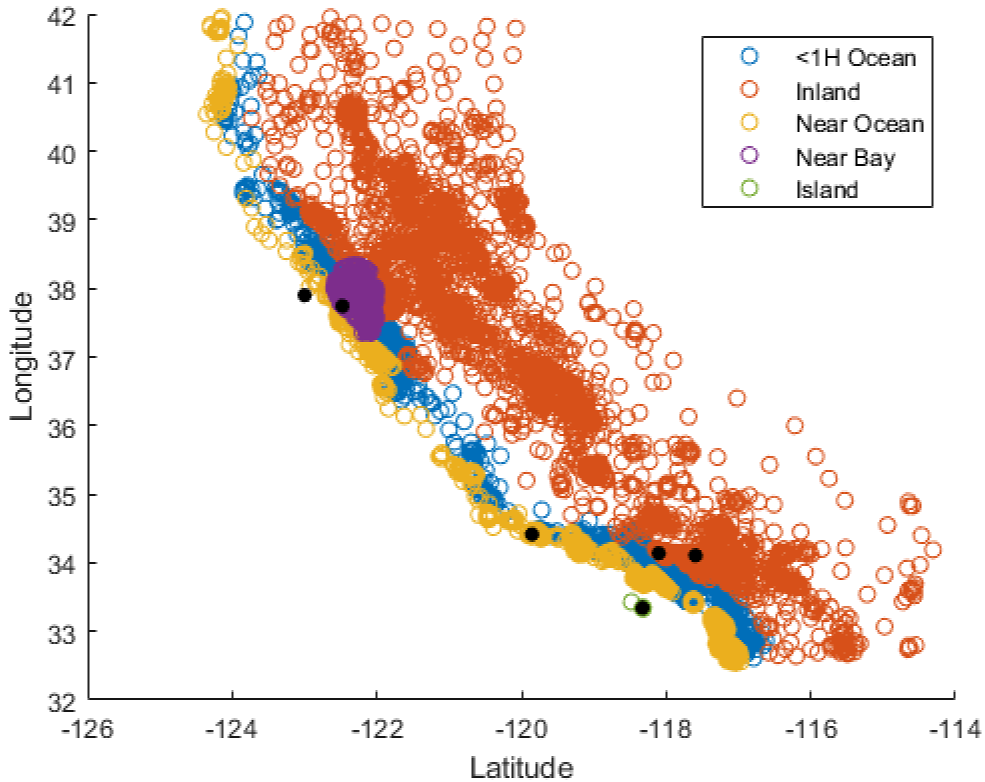
20×3 table

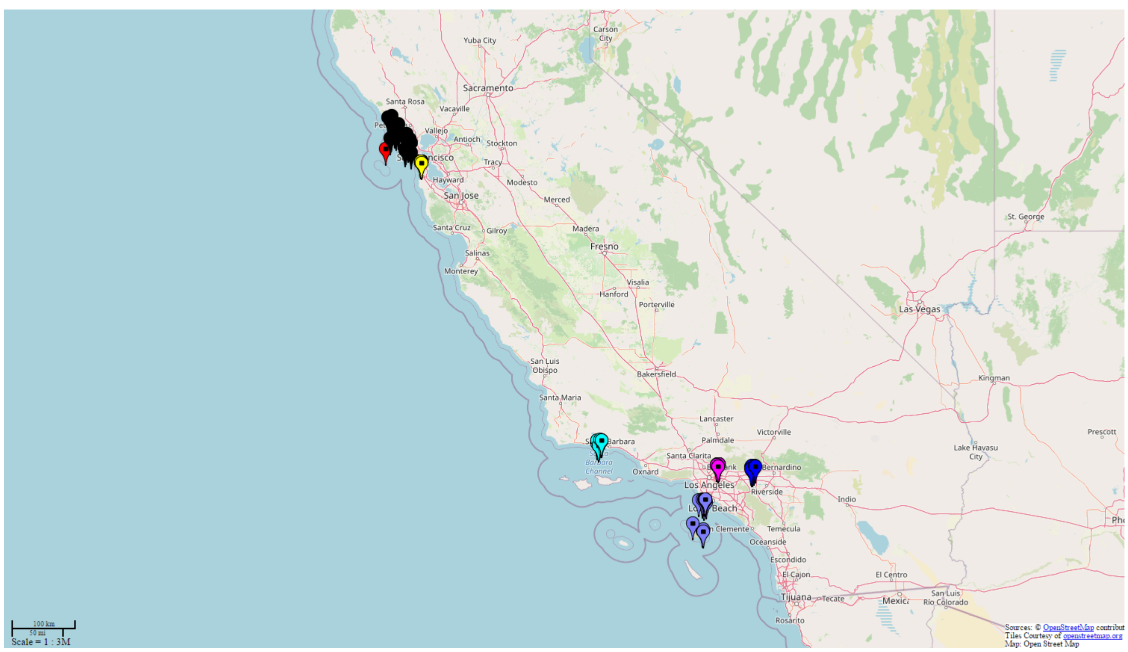
Longitude	Latitude	HousingClass
-118.33	33.34	5
-118.32	33.34	5
-118.32	33.35	5
-118.32	33.33	5
-118.48	33.43	5
-118.31	33.67	3
-118.28	33.68	3
-118.33	33.69	3
-118.29	33.71	3
-118.29	33.71	3
-118.29	33.71	3
-118.29	33.71	3
-118.39	33.71	3
-118.33	33.72	3
-118.31	33.72	3
-118.3	33.72	3
-118.3	33.72	3
-118.3	33.72	3
-118.29	33.72	3
-118.29	33.72	3

Webmap

```
wm = webmap('Open Street Map');
newhouses = geopoint(x2new, x1new);
webmarker_nh = wmmarker(newhouses, 'Color', 'red');

h1 = geopoint(house1x2, house1x1);
h2 = geopoint(house2x2, house2x1);
h3 = geopoint(house3x2, house3x1);
h4 = geopoint(house4x2, house4x1);
h5 = geopoint(house5x2, house5x1);
h6 = geopoint(house6x2, house6x1);

wmmarker(h1, 'Color', 'b');
wmmarker(h2, 'Color', 'k');
wmmarker(h3, 'Color', 'y');
wmmarker(h4, 'Color', 'm');
wmmarker(h5, 'Color', 'c');
wmmarker(h6, 'Color', [0.5 0.5 1]);
```



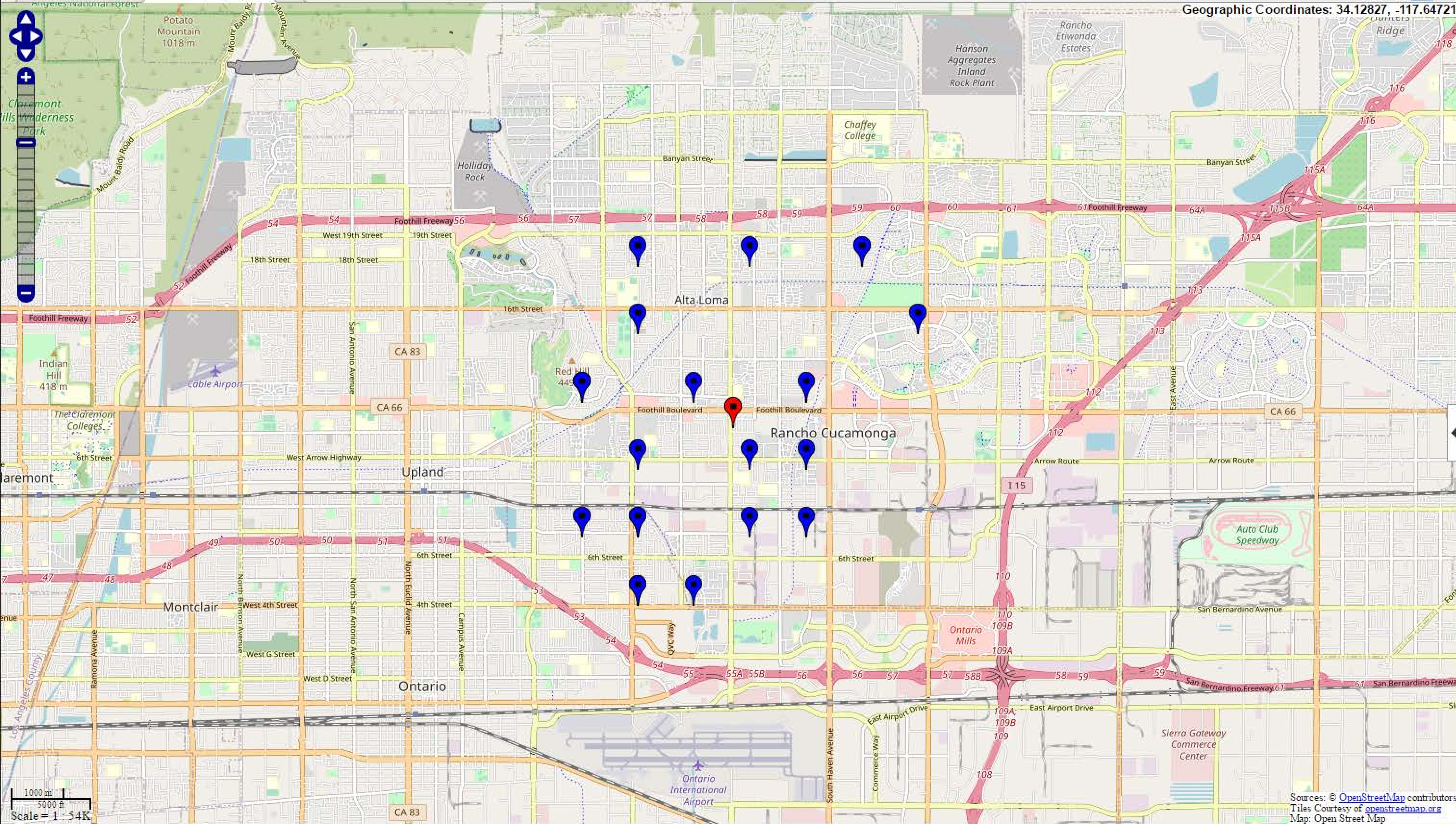


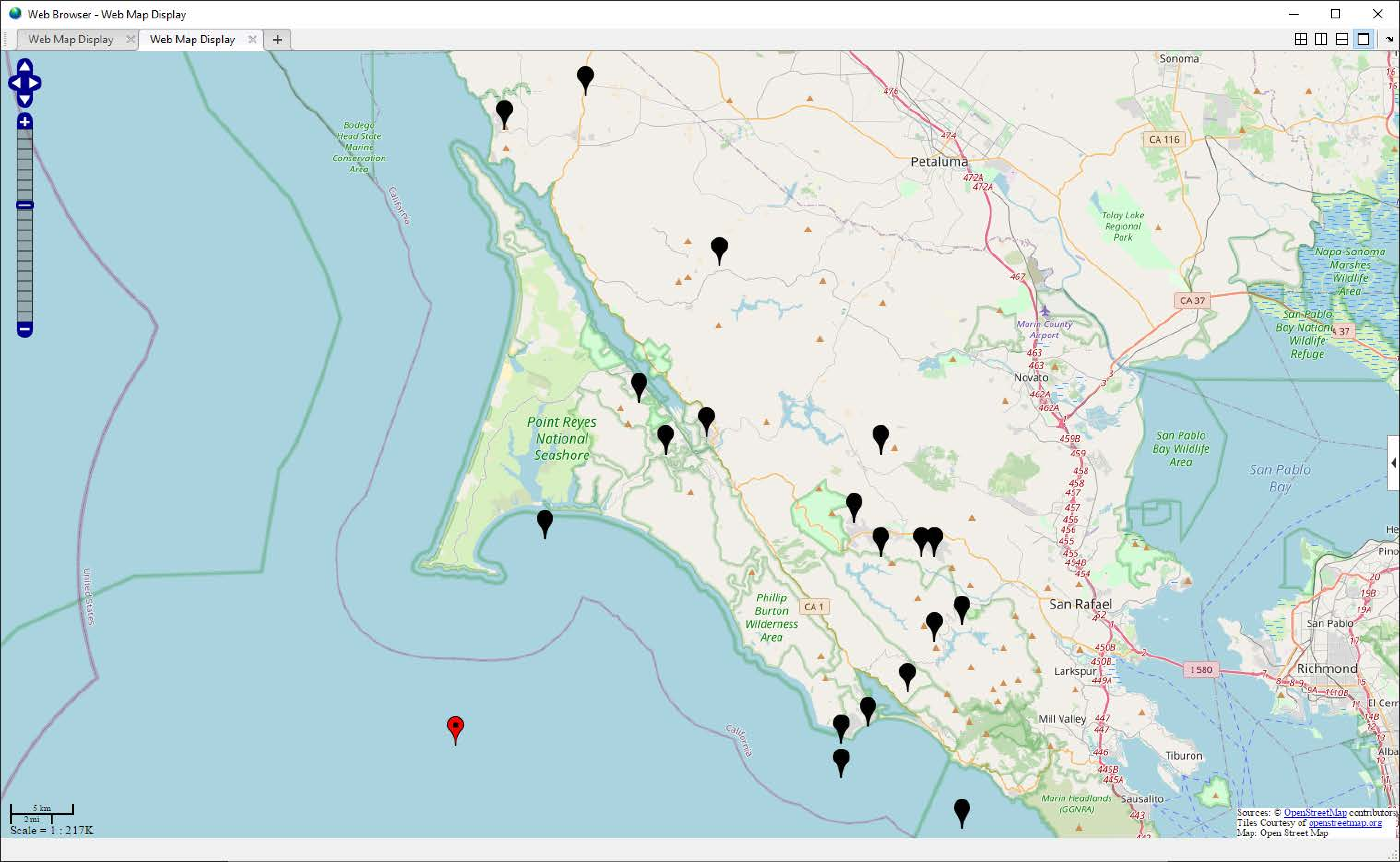


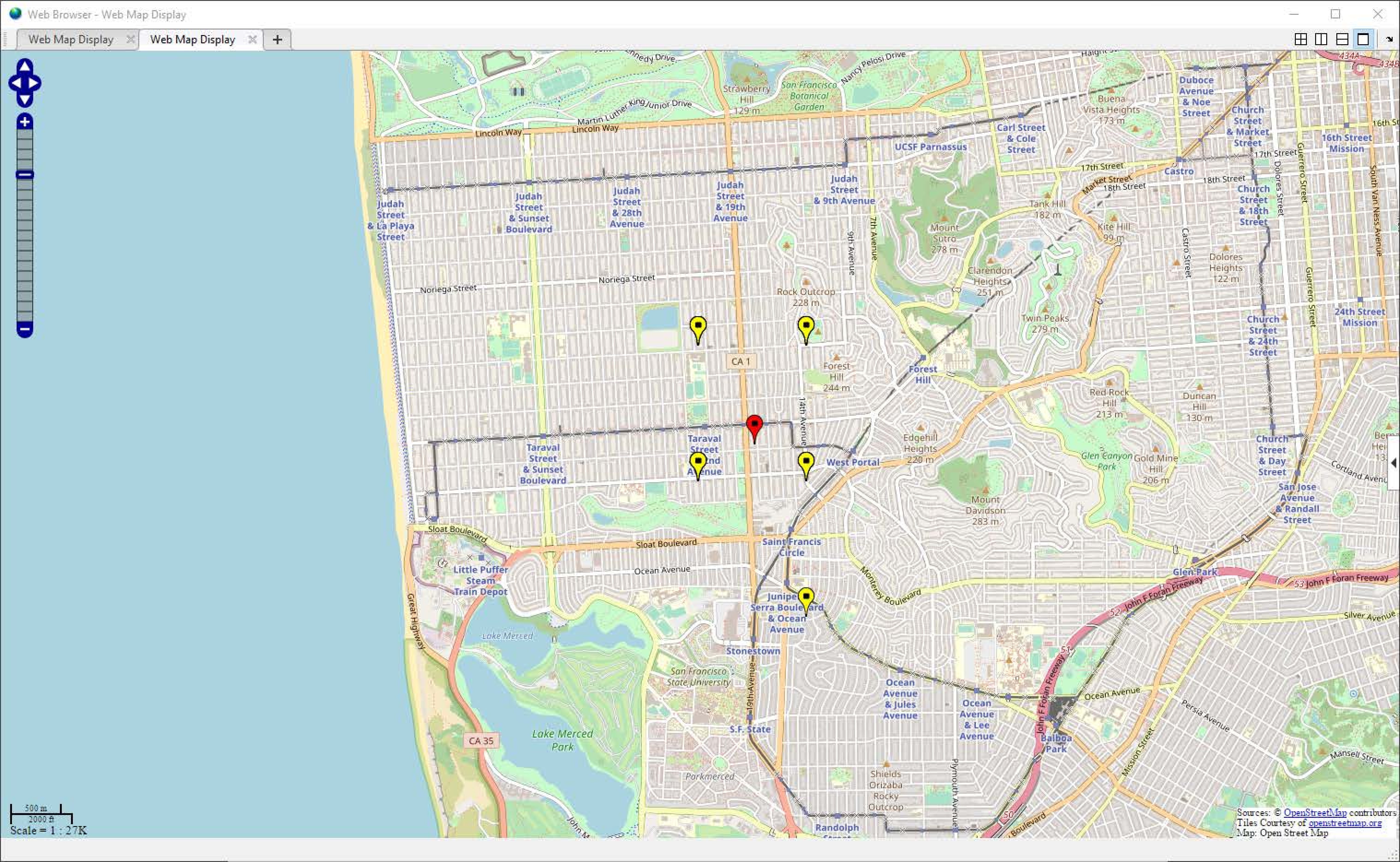
Published with MATLAB® R2018b

Web Browser - Web Map Display

Web Map Display | Web Map Display | +

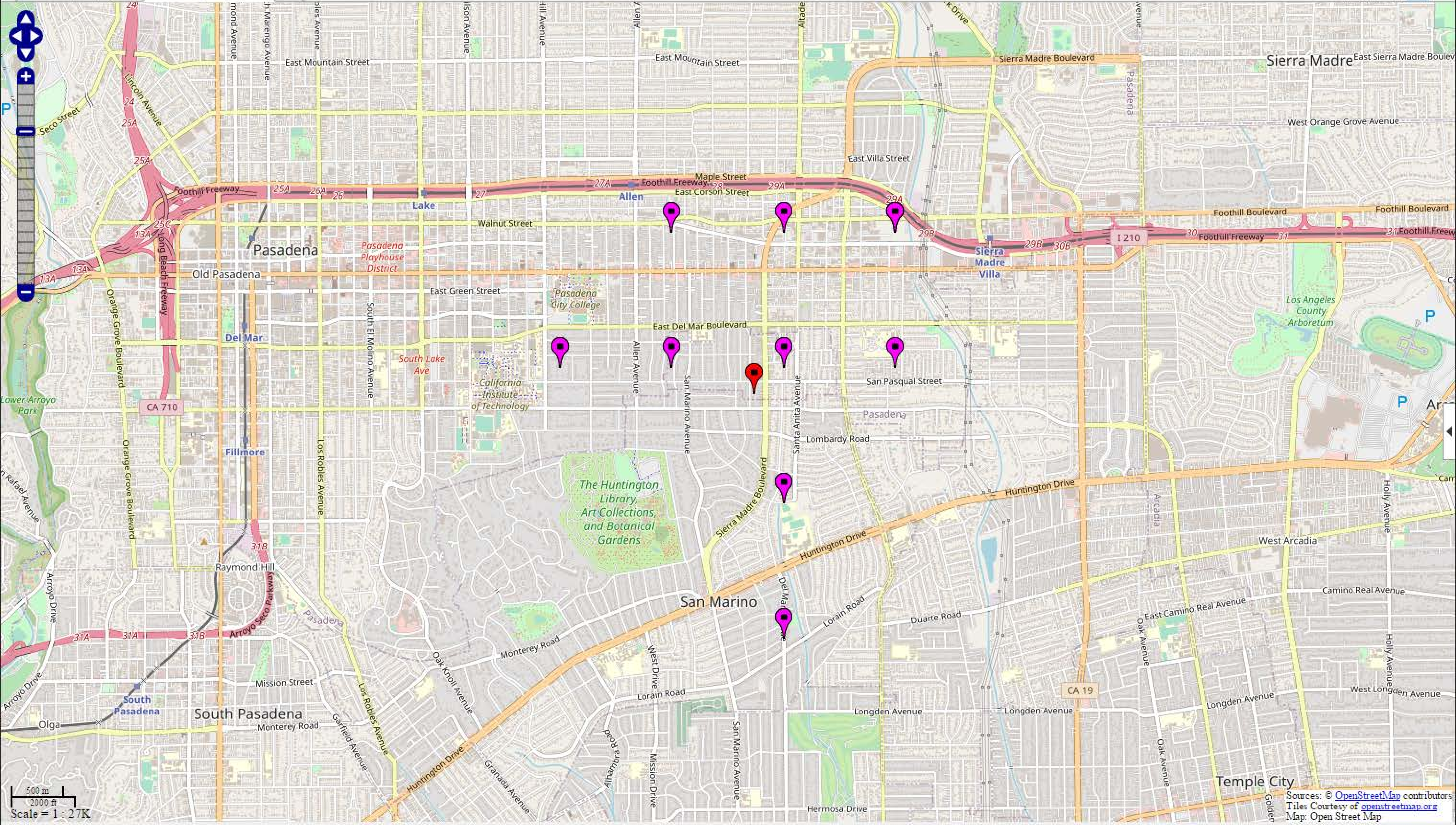


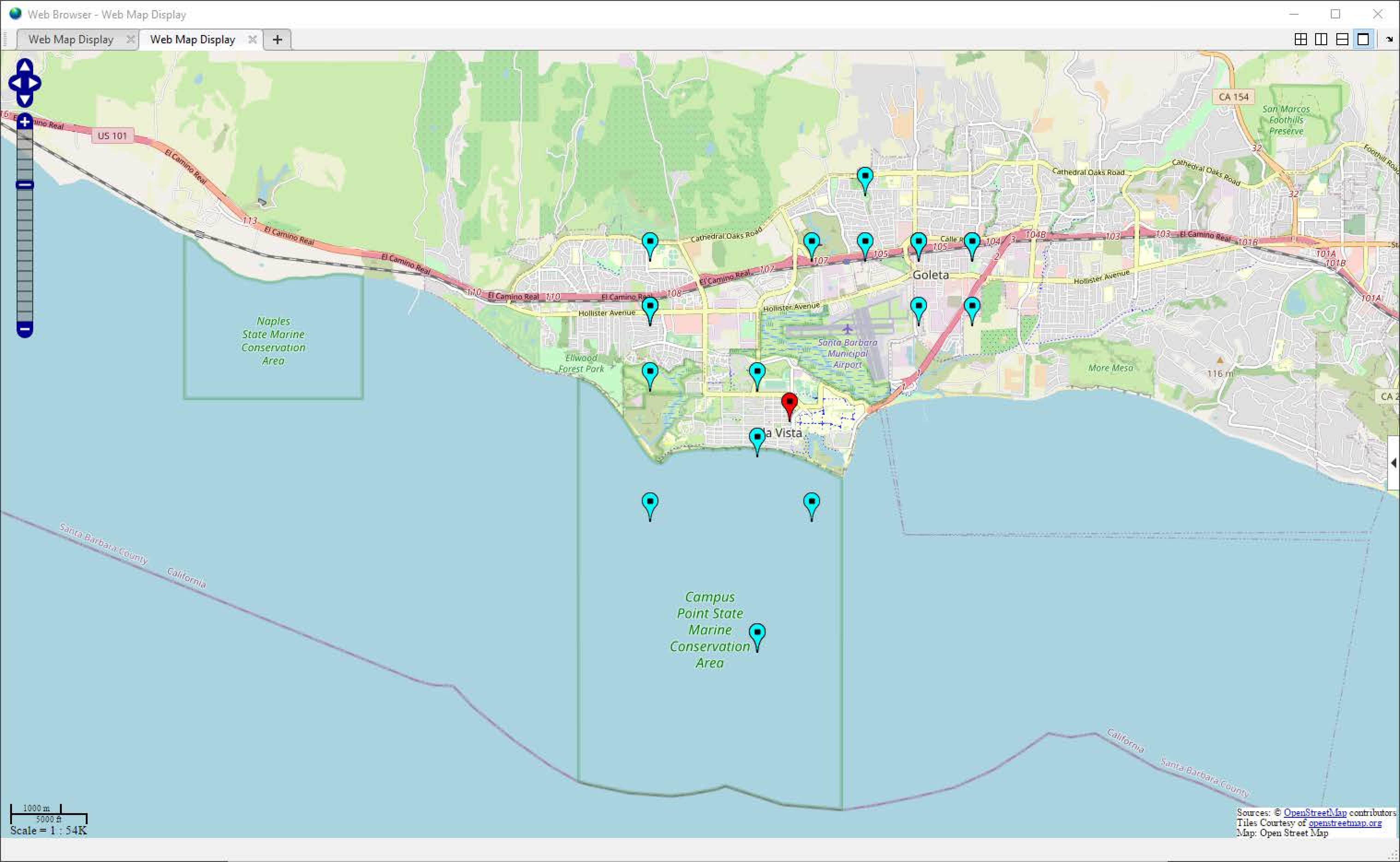

Web Browser - Web Map Display
Web Map Display
Web Map Display
Sonoma
Petaluma
CA 116
Bodega Head State Marine Conservation Area
California
Tolay Lake Regional Park
Napa-Sonoma Marshes Wildlife Area
CA 37
Marin County Airport
San Pablo Bay National Wildlife Refuge
Novato
Point Reyes National Seashore
San Pablo Bay Wildlife Area
San Pablo Bay
Phillip Burton Wilderness Area
CA 1
San Rafael
San Pablo
Richmond
I 580
Larkspur
Mill Valley
Tiburon
Sausalito
Marin Headlands (GGNRA)
United States
California
5 km
2 mi
Scale = 1 : 217K
Sources: © OpenStreetMap contributors
Tiles Courtesy of openstreetmap.org
Map: Open Street Map

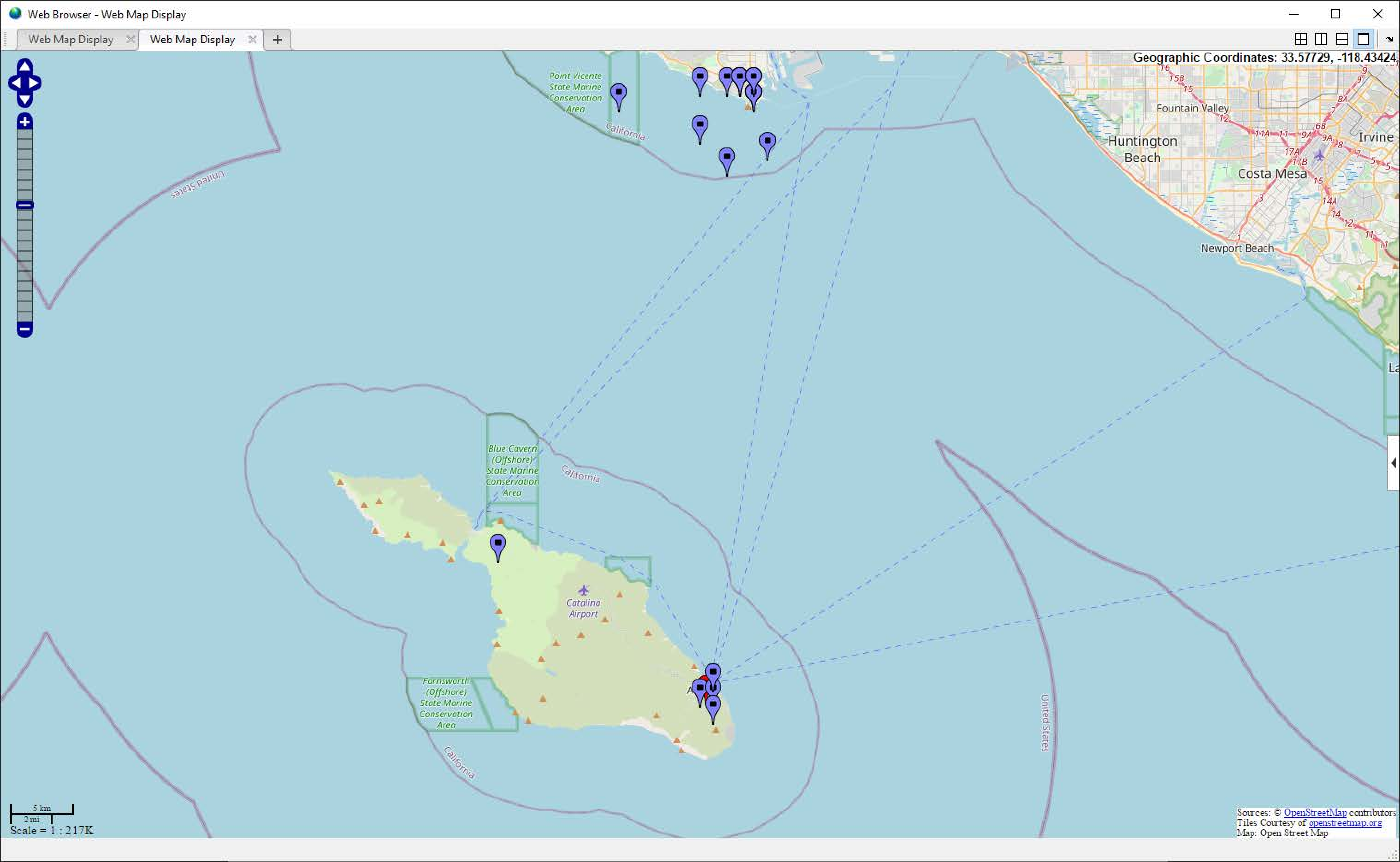

Web Browser - Web Map Display
Web Map Display
Web Map Display
Lincoln Way
Martin Luther King Junior Drive
Strawberry Hill 129 m
San Francisco Botanical Garden
Nancy Pelosi Drive
Judah Street & La Playa Street
Judah Street & Sunset Boulevard
Judah Street & 28th Avenue
Judah Street & 19th Avenue
Judah Street & 9th Avenue
UCSF Parnassus
Carl Street & Cole Street
Buena Vista Heights 173 m
Duboce Avenue & Noe Street
Church Street & Market Street
16th Street Mission
17th Street
18th Street
Market Street
Castro
Church Street & 18th Street
Tank Hill 182 m
Kite Hill 99 m
Mount Sutro 278 m
Clarendon Heights 251 m
Dolores Heights 122 m
Castro Street
Dolores Street
Guerrero Street
South Van Ness Avenue
Noriega Street
Rock Outcrop 228 m
Twin Peaks 279 m
24th Street Mission
Church Street & 24th Street
CA 1
Forest Hill 244 m
Forest Hill
Red Rock Hill 213 m
Duncan Hill 130 m
9th Avenue
7th Avenue
14th Avenue
Taraval Street & Sunset Boulevard
Taraval Street & 19th Avenue
West Portal
Edgehill Heights 220 m
Glen Canyon Park
Gold Mine Hill 206 m
Church Street & Day Street
San Jose Avenue & Randall Street
Cortland Avenue
Mount Davidson 283 m
Sloat Boulevard
Saint Francis Circle
Ocean Avenue
Little Puffer Steam Train Depot
Monterey Boulevard
Glen Park
John F Foran Freeway
Great Highway
Junipero Serra Boulevard & Ocean Avenue
Silver Avenue
Lake Merced
Stonestown
San Francisco State University
19th Avenue
Ocean Avenue & Jules Avenue
Ocean Avenue & Lee Avenue
Balboa Park
Persia Avenue
Mission Street
Mansell Street
CA 35
Lake Merced Park
S.F. State
Parkmerced
Shields Orizaba Rocky Outcrop
Plymouth Avenue
Randolph Street
500 m
2000 ft
Scale = 1 : 27K
Sources: © OpenStreetMap contributors
Tiles Courtesy of openstreetmap.org
Map: Open Street Map

Web Browser - Web Map Display

Web Map Display | Web Map Display




Web Browser - Web Map Display
Web Map Display
Web Map Display
US 101
El Camino Real
Cathedral Oaks Road
Hollister Avenue
Goleta
CA 154
San Marcos Foothills Preserve
Naples State Marine Conservation Area
Ellwood Forest Park
Santa Barbara Municipal Airport
More Mesa
116 m
Campus Point State Marine Conservation Area
Santa Barbara County
California
1000 m
5000 ft
Scale = 1 : 54K
Sources: © OpenStreetMap contributors
Tiles Courtesy of openstreetmap.org
Map: Open Street Map



Web Browser - Web Map Display
Web Map Display
Web Map Display
Geographic Coordinates: 33.57729, -118.43424
Point Vicente State Marine Conservation Area
California
Fountain Valley
Huntington Beach
Irvine
Costa Mesa
Newport Beach
United States
Blue Cavern (Offshore) State Marine Conservation Area
California
Catalina Airport
Farnsworth (Offshore) State Marine Conservation Area
California
United States
5 km
2 mi
Scale = 1 : 217K
Sources: © OpenStreetMap contributors
Tiles Courtesy of openstreetmap.org
Map: Open Street Map

Table of Contents

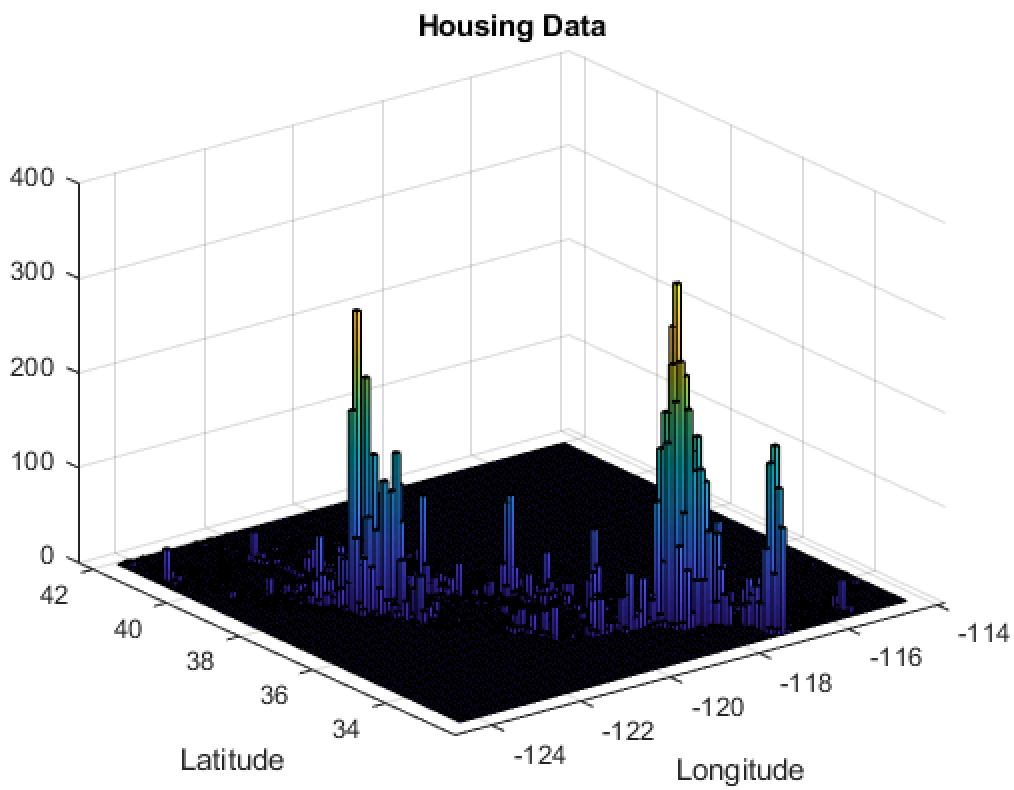
BE606 HW3 Problem 2a

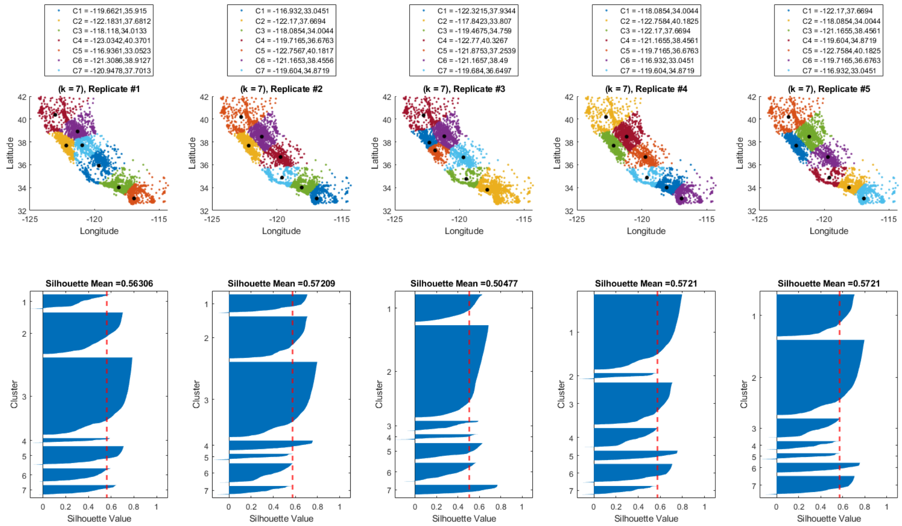
```
clear all
close all
```

Part 2

```
A = readtable('housing.csv');
B = table2array(A(:,1:9));
x1 = B(:,1);
x2 = B(:,2);
X = [x1,x2];


figure;
hist3(X,'CDataMode','auto','FaceColor','interp','Nbins',[100 100])
title('Housing Data')
xlabel('Longitude')
ylabel('Latitude')
% tic
%kmeans
f = figure;

for rr = 1:5
    [class,cent] = kmeans(X,7);
    subplot(2,5,rr)
%      tic
    for kk = 1:7
        hold on
        plot(x1(class==kk),x2(class==kk),'.','DisplayName',...
            ['C',num2str(kk),' =
 ',num2str(cent(kk,1)),',',num2str(cent(kk,2))])
        legend('Location', 'northoutside')


 plot(cent(kk,1),cent(kk,2),'.','MarkerSize',15,'color','k', 'HandleVisibility','o

    end
%      toc

    hold off
    title(['(k = 7), Replicate #', num2str(rr)])
    xlabel('Longitude')
```

```
    ylabel('Latitude')

    subplot(2,5,rr+5)
    sil = silhouette(X,class, 'Euclidean'); %save value for mean
%     toc
    silhouette(X,class, 'Euclidean') %repeat to easily plot
%     toc
    hold on
    xline(mean(sil), 'r--', 'LineWidth', 2);
    hold off

    title(['Silhouette Mean =', num2str(mean(sil))])
end
% toc
f.WindowState = 'maximized'; %use this to maximize plot on screen
%though it does not work so well in the printout
```





Part 3 Questions

```
disp('Q1')
disp('Clusters 2 and 3 of replicate 2 contain the largest population
 centers in California. Cluster 2 contains LA County, the largest
 county in the US')
disp('and cluster 3 contains the bay area.')
disp('Q2')
disp('Yes the 7 clusters make sense, each cluter either contains
 population centers, or the distance is minimized. Centroid 6 is
 Northern California, where there are mainly national forests and some
 smaller cities/towns')
disp('Centroid 2 contains LA County, largest population center in the
 state')
disp('Centroid 3 contains the Bay area and its most likely commuting
 areas')
disp('Centroid 4 is essentially San Diego and its commuting regions,
 as well as reservations.')
disp('Centroid 5 is mostly Yosemite and other forests in the
 interior')
disp('Centroid 1 is similar to centroid 5, with the inclusion of
 Sacramento')
disp('Q3')
disp('Between 2 and 3, there is a decrease in distance and essentially
 a split of LA County, and an increase in the lower middle centroid.')
disp('Q4')
disp('The equation for silhouette coeff, uses max distances in its
 denominator, and since the middle centroid increased in size, and
 pushed two centroids close together, we see a reduction in overall
 silhouette score.')
disp('Q5')
disp('I prefer replicate 2, as it still has a high silhouette score,
 and pretty accurately depicts the Bay Area, LA County, San Diego
```

```
as well as the national forests/parks. There should be a few large
 clusters, since there are higher population densities in certain
 regions.')
```

Q1
Clusters 2 and 3 of replicate 2 contain the largest population centers in California. Cluster 2 contains LA County, the largest county in the US
and cluster 3 contains the bay area.
Q2
Yes the 7 clusters make sense, each cluter either contains population centers, or the distance is minimized. Centroid 6 is Northern California, where there are mainly national forests and some smaller cities/towns
Centroid 2 contains LA County, largest population center in the state
Centroid 3 contains the Bay area and its most likely commuting areas
Centroid 4 is essentially San Diego and its commuting regions, as well as reservations.
Centroid 5 is mostly Yosemite and other forests in the interior
Centroid 1 is similar to centroid 5, with the inclusion of Sacramento
Q3
Between 2 and 3, there is a decrease in distance and essentially a split of LA County, and an increase in the lower middle centroid.
Q4
The equation for silhouette coeff, uses max distances in its denominator, and since the middle centroid increased in size, and pushed two centroids close together, we see a reduction in overall silhouette score.
Q5
I prefer replicate 2, as it still has a high silhouette score, and pretty accurately depicts the Bay Area, LA County, San Diego as well as the national forests/parks. There should be a few large clusters, since there are higher population densities in certain regions.

Published with MATLAB® R2018b

Table of Contents

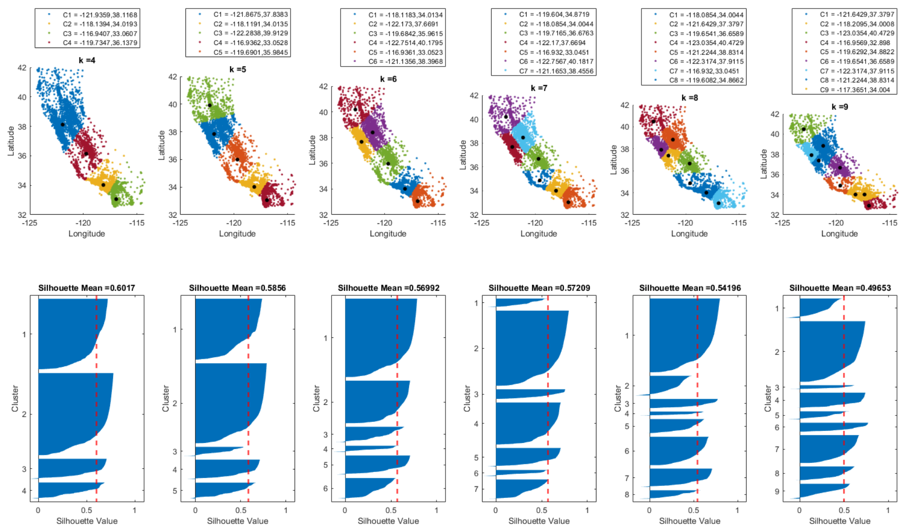
BE606 HW3 Problem 2b

```
clear all
close all
```

Part 2

```
A = readtable('housing.csv');
B = table2array(A(:,1:9));
x1 = B(:,1);
x2 = B(:,2);
X = [x1,x2];



f = figure;
for kk = 4:1:9
    [class,cent] = kmeans(X,kk,'Replicates',100);
    subplot(2,6,kk-3)

    for jj = 1:kk
        hold on

        plot(x1(class==jj),x2(class==jj),'.','DisplayName',...
            ['C',num2str(jj),' =
 ',num2str(cent(jj,1)),',',num2str(cent(jj,2))])
        legend('Location', 'northoutside')


 plot(cent(jj,1),cent(jj,2),'.','MarkerSize',15,'color','k', 'HandleVisibility','o

    end
    hold off

    title(['k =', num2str(kk)])
    xlabel('Longitude')
    ylabel('Latitude')


    subplot(2,6,kk+3)
    sil = silhouette(X,class, 'Euclidean'); %save value for mean

    silhouette(X,class, 'Euclidean') %repeat to easily plot

    hold on
```

```
    xline(mean(sil), 'r--', 'LineWidth', 2);
    hold off
    title(['Silhouette Mean =', num2str(mean(sil))])
end
f.WindowState = 'maximized';
```



Part 3 Questions

```
disp('Q1')
disp('The two large clusters contain where the majority of the
 population lives. One contains LA County and its surroundings, and
 the other contains nearly a quarter of California.')
disp('Q2')
disp('In k=5, there is a marked improvement as the large northern
 California cluster can be split in two, allowing for more appropriate
 geographical classification. San Francisco is now in its own
 cluster.')
disp('Q3')
disp('Yes, the silhouette score did not decrease much, and now each
 population center in CA is more accurately described. Cluster size is
 also not as drastically large.')
disp('Q4')
disp('After k=7 the silhouette score begins to decrease. Though there
 are more clusters, sectioning off cities and communities possibly
 more effectively, we are beginning to generate too many clusters.
 Realistically the two best were k=6/7 considering the silhouette
 mean is relatively close, and the map intuitively maps the major
 population centers.')
```

Q1

The two large clusters contain where the majority of the population lives. One contains LA County and its surroundings, and the other contains nearly a quarter of California.

Q2
In k=5, there is a marked improvement as the large northern California cluster can be split in two, allowing for more appropriate geographical classification. San Francisco is now in its own cluster.
Q3
Yes, the silhouette score did not decrease much, and now each population center in CA is more accurately described. Cluster size is also not as drastically large.
Q4
After k=7 the silhouette score begins to decrease. Though there are more clusters, sectioning off cities and communities possibly more effectively, we are beginning to generate too many clusters. Realistically the two best were k=6/7 considering the silhouette mean is relatively close, and the map intuitively maps the major population centers.

Published with MATLAB® R2018b

Table of Contents

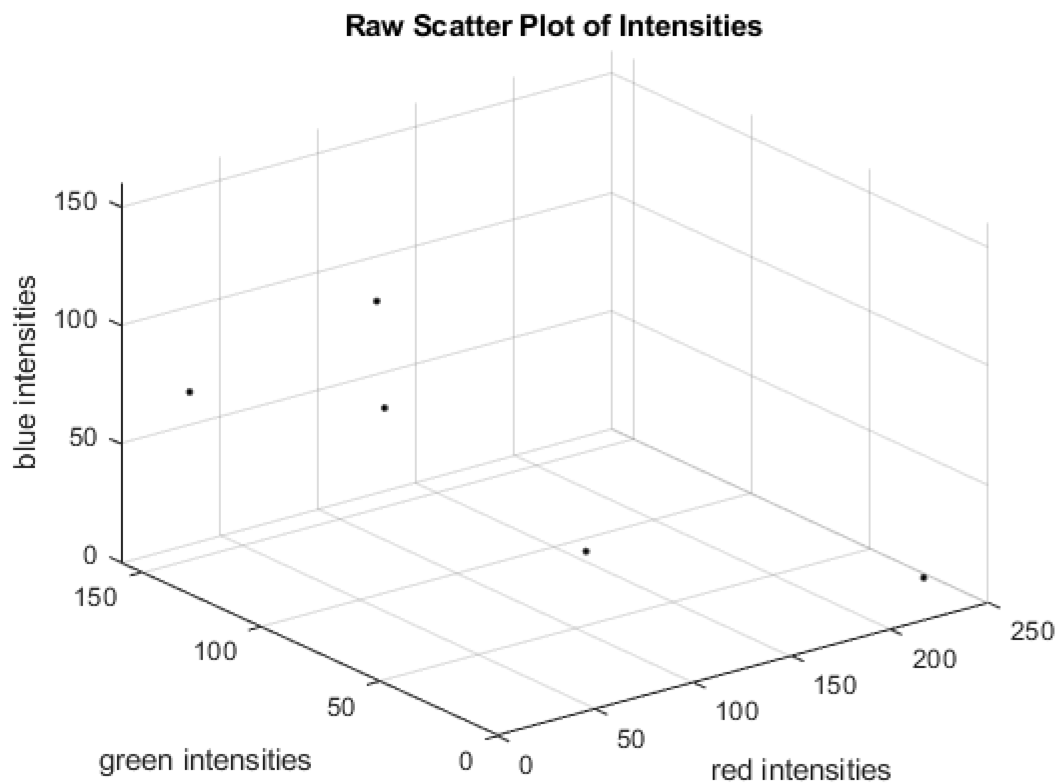
BE606 HW3 Problem 2c

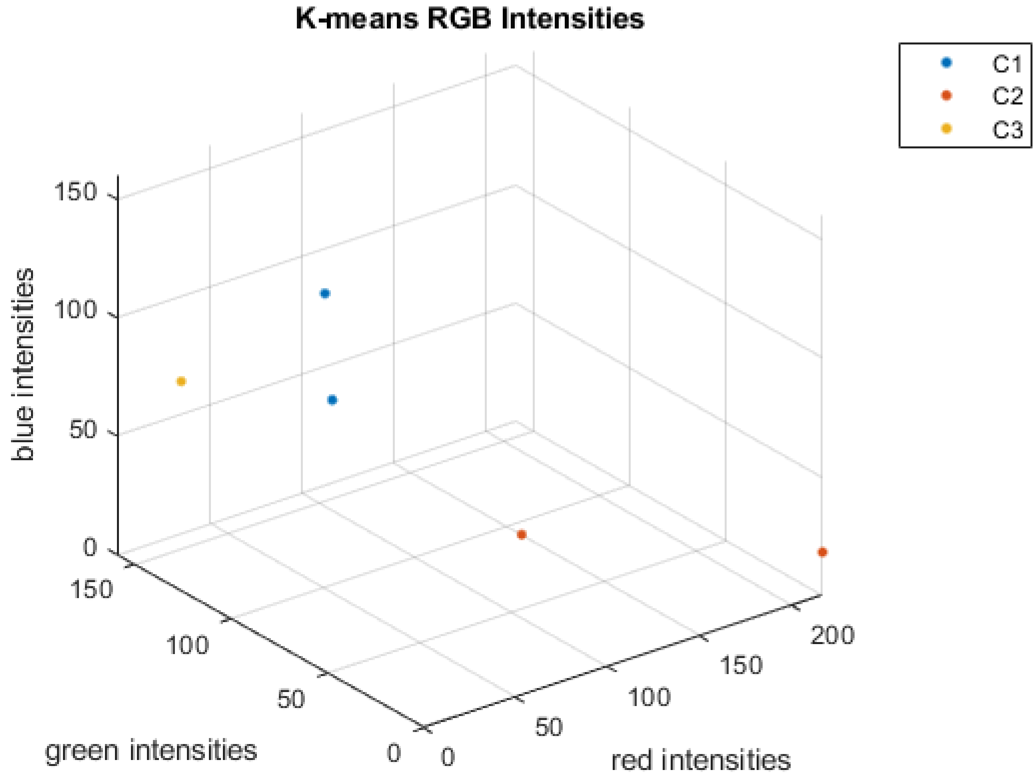
```
clear all
close all
```

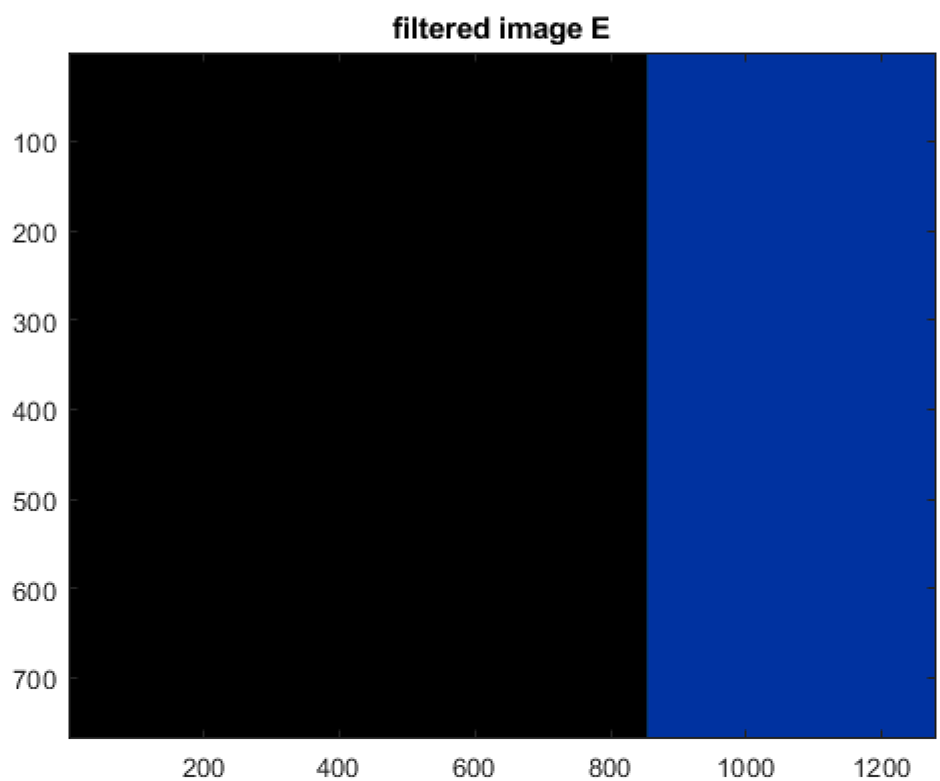
Part 1

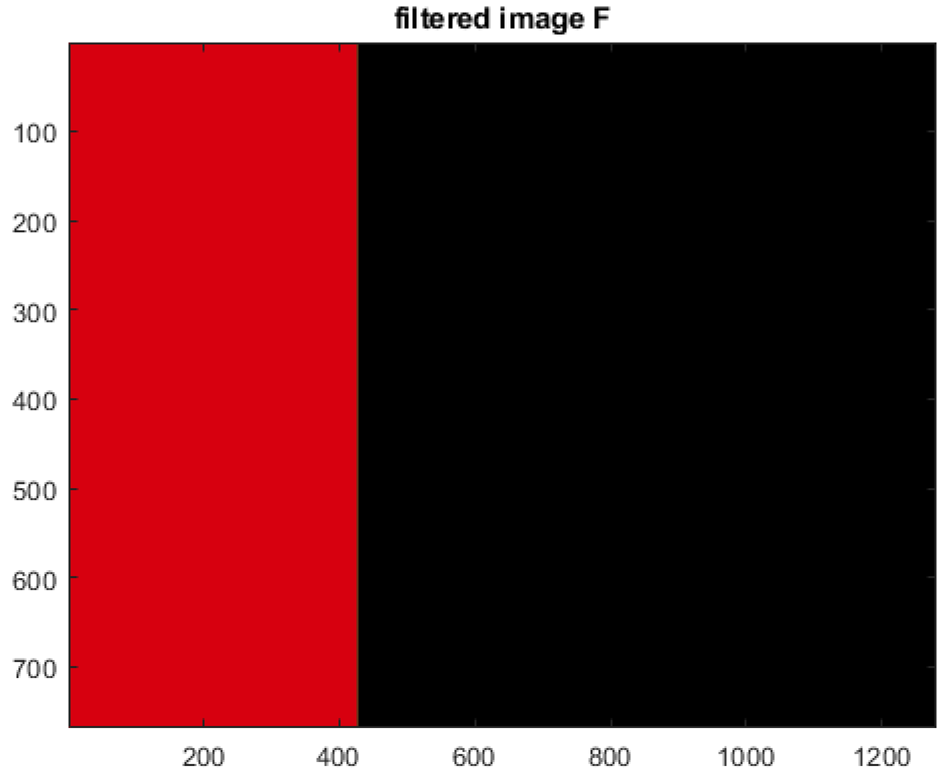
```
A = double(imread('default_rgb_reference.tif'));

red = A(:,:,1);
green = A(:,:,2);
blue = A(:,:,3);

figure;
plot3(red,green,blue, 'k.')
grid on
title('Raw Scatter Plot of Intensities')

xlabel('red intensities')
ylabel('green intensities')
zlabel('blue intensities')


X = [red(:), green(:), blue(:)];


[class,cent] = kmeans(X,3,'Replicates',10);

figure;

for i = 1:3
    cl = i == class;
    plot3(red(cl),green(cl),blue(cl),'.','MarkerSize',12)
    hold on
end
grid on
legend('C1','C2','C3')
title('K-means RGB Intensities')
xlabel('red intensities')
ylabel('green intensities')
zlabel('blue intensities')
```

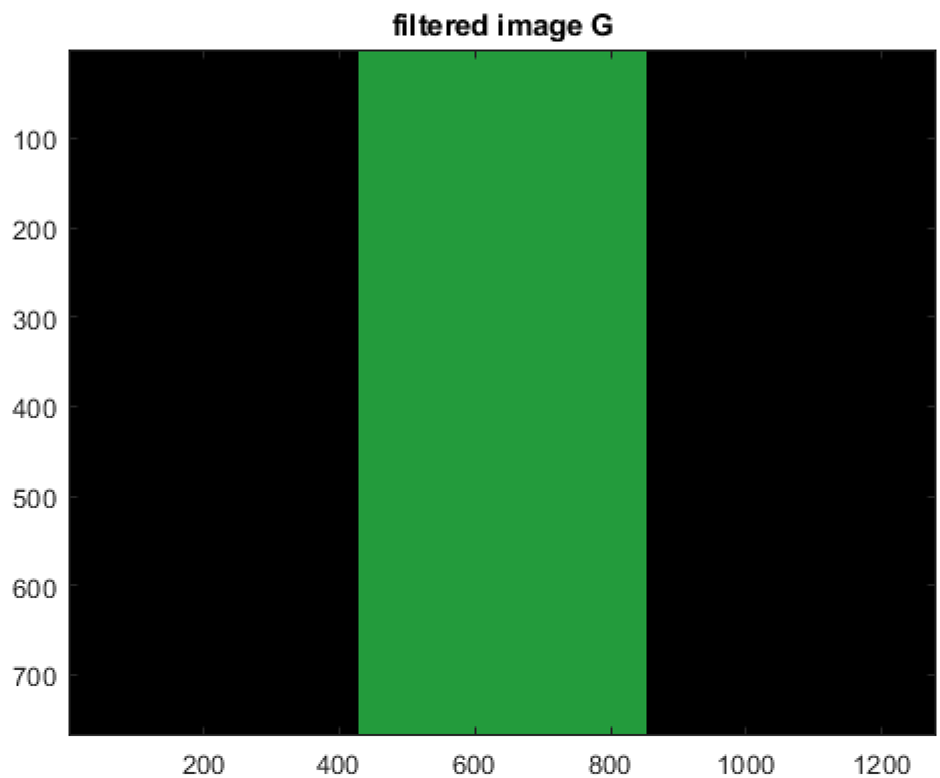
```
E = NaN*ones(size(X));
E(1==class,:) = X(1==class,:);
E = uint8(reshape(E,size(A)));
figure;
image(E)
title('filtered image E')


F = NaN*ones(size(X));
F(2==class,:) = X(2==class,:);
F = uint8(reshape(F,size(A)));
figure;
image(F)
title('filtered image F')

G = NaN*ones(size(X));
G(3==class,:) = X(3==class,:);
G = uint8(reshape(G,size(A)));
figure;
image(G)
title('filtered image G')
```




K-means RGB Intensities
C1
C2
C3
blue intensities
green intensities
red intensities
0
50
100
150
200


filtered image E
100
200
300
400
500
600
700
200
400
600
800
1000
1200

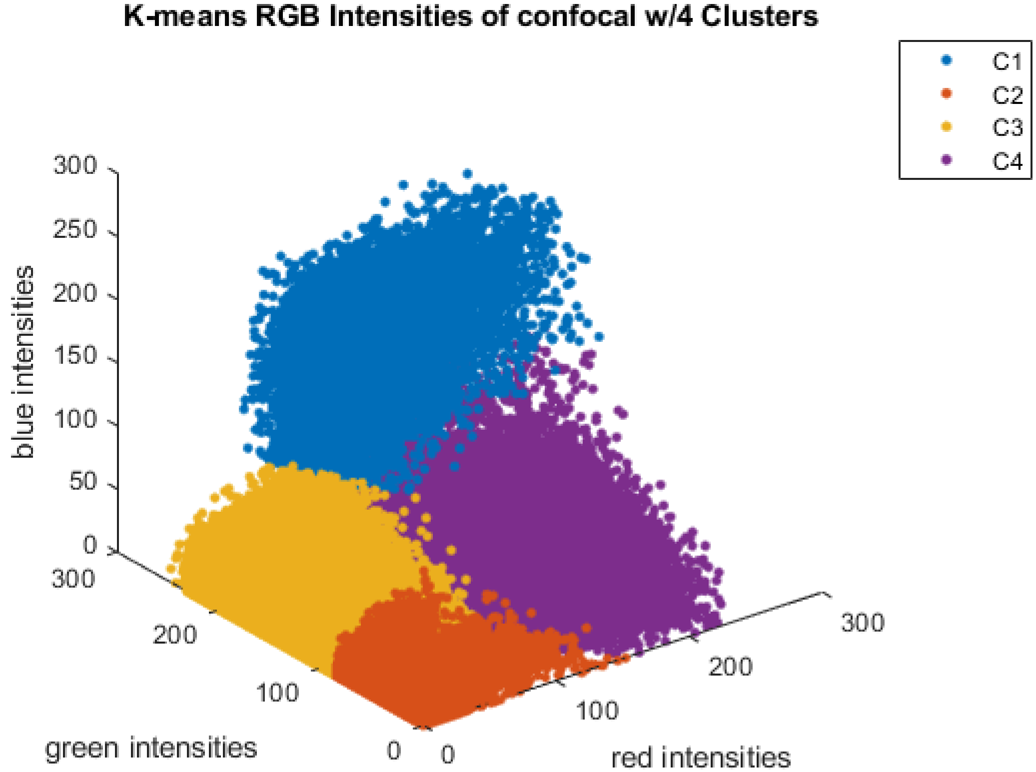

filtered image F
100
200
300
400
500
600
700
200
400
600
800
1000
1200

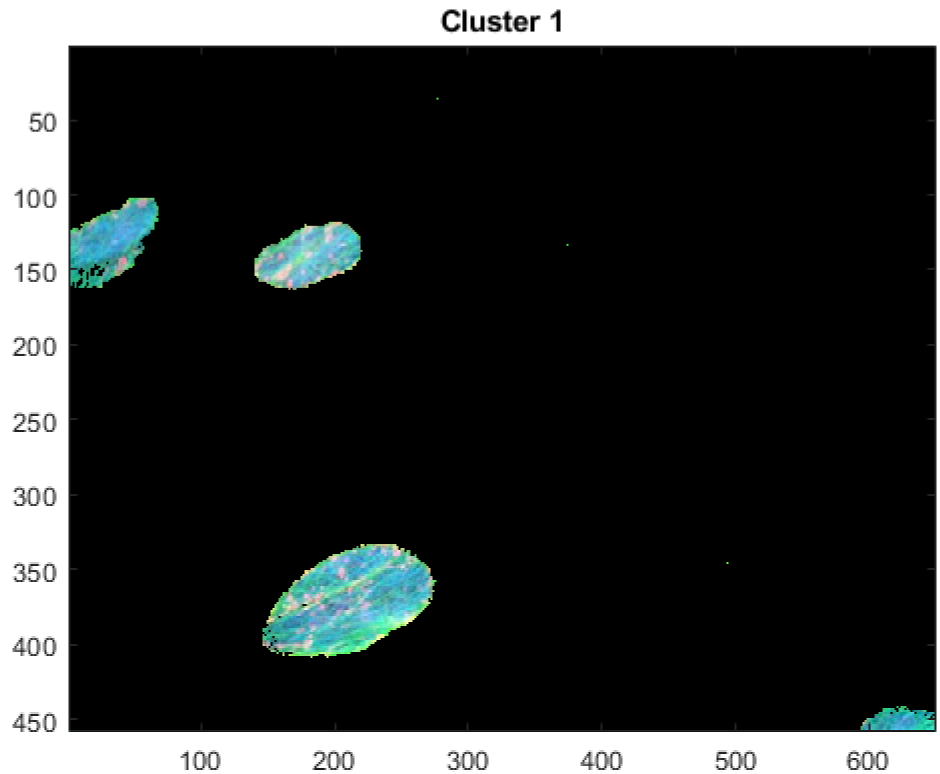

filtered image G
100
200
300
400
500
600
700
200
400
600
800
1000
1200

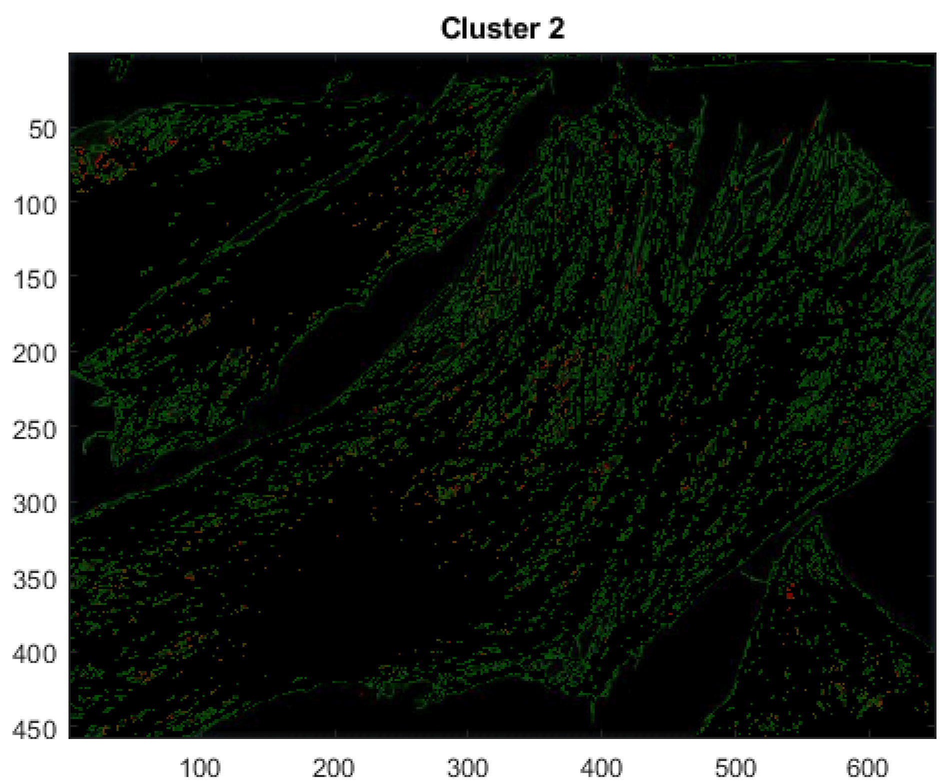
Part 2

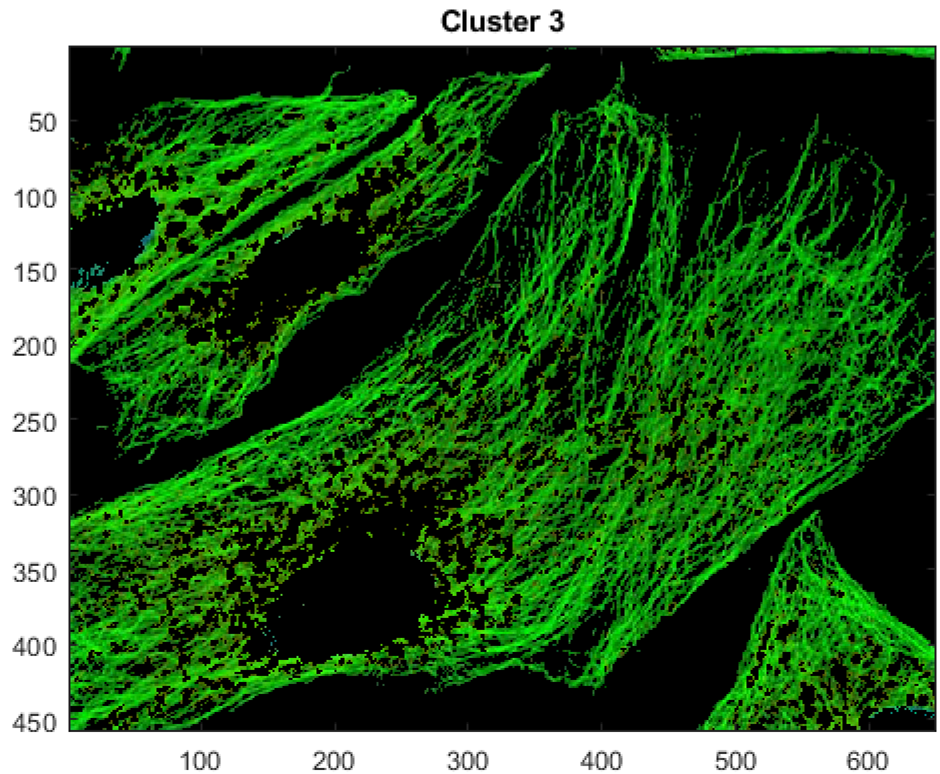
```
A = double(imread('confocal_image01.tif'));

red = A(:,:,1);
green = A(:,:,2);
blue = A(:,:,3);

X = [red(:), green(:), blue(:)];
%Using 4 centroids intuitively makes sense here--we have R,G,B and
 Black
%colors in the image
[class,cent] = kmeans(X,4,'Replicates',50);

figure;

for i = 1:4
    cl = i == class;
    plot3(red(cl),green(cl),blue(cl),'.','MarkerSize',12)
    hold on
end
legend('C1','C2','C3','C4')
title('K-means RGB Intensities of confocal w/4 Clusters')
xlabel('red intensities')
ylabel('green intensities')
zlabel('blue intensities')


E = NaN*ones(size(X));
E(1==class,:) = X(1==class,:);
E = uint8(reshape(E,size(A)));
figure;
image(E)
title('Cluster 1')


F = NaN*ones(size(X));
F(2==class,:) = X(2==class,:);
F = uint8(reshape(F,size(A)));
figure;
image(F)
title('Cluster 2')

G = NaN*ones(size(X));
G(3==class,:) = X(3==class,:);
G = uint8(reshape(G,size(A)));
figure;
image(G)
title('Cluster 3')


H = NaN*ones(size(X));
H(4==class,:) = X(4==class,:);
```

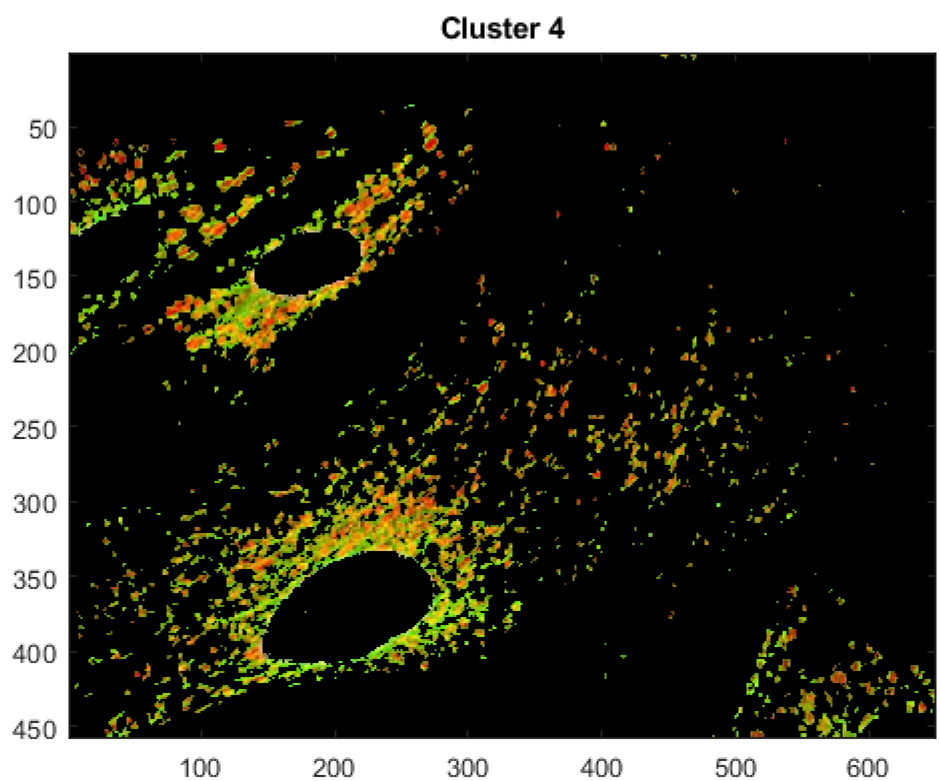
```
H = uint8(reshape(H,size(A)));
figure;
image(H)
title('Cluster 4')
```




Cluster 1
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600

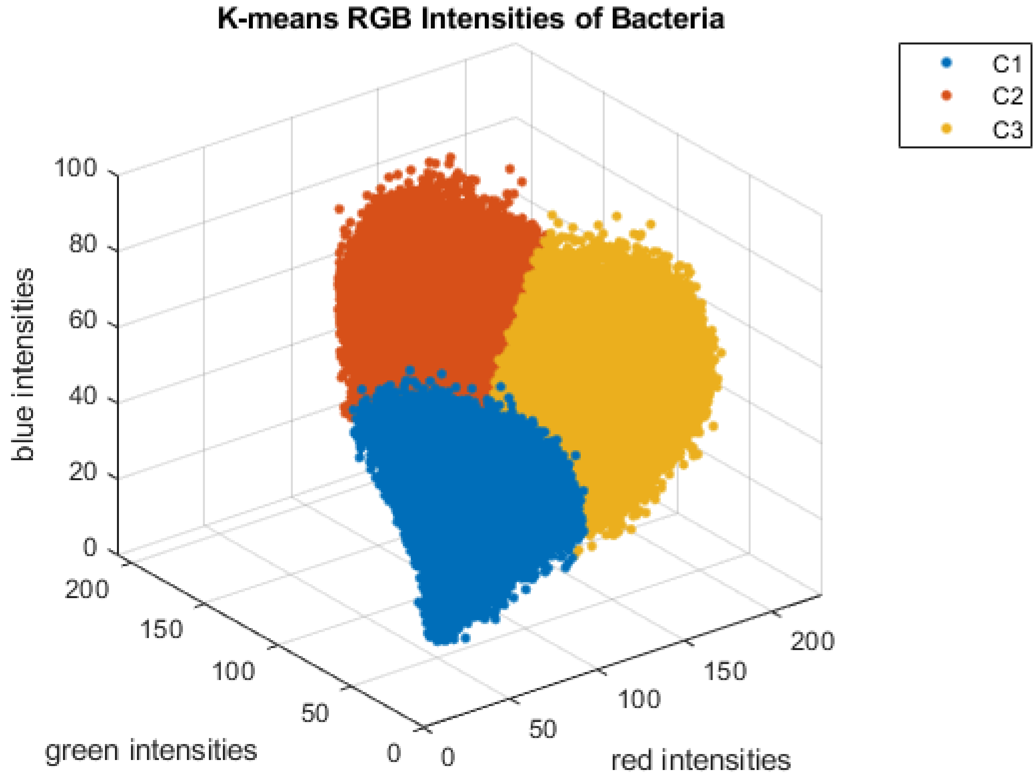

Cluster 2
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600

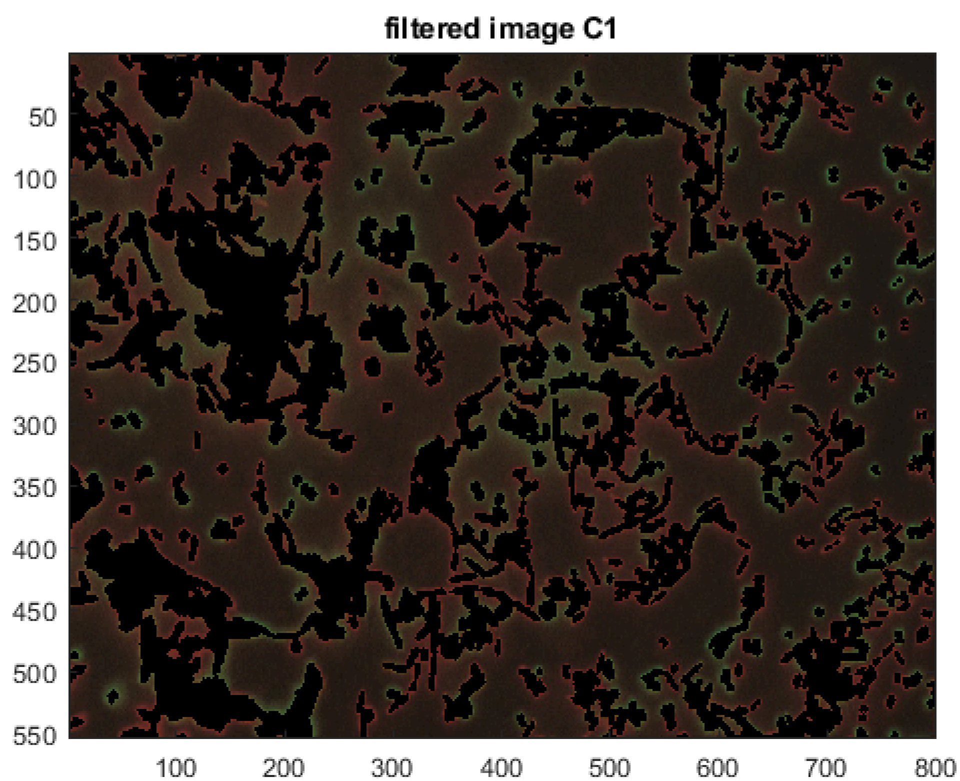

Cluster 3
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600

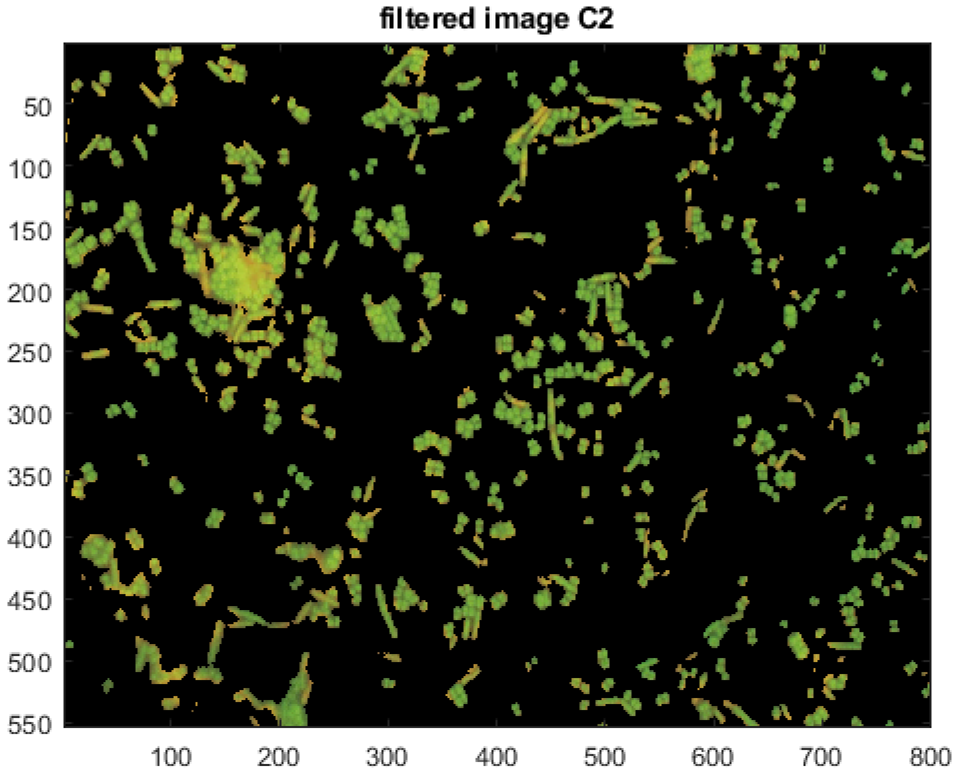

Cluster 4
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600

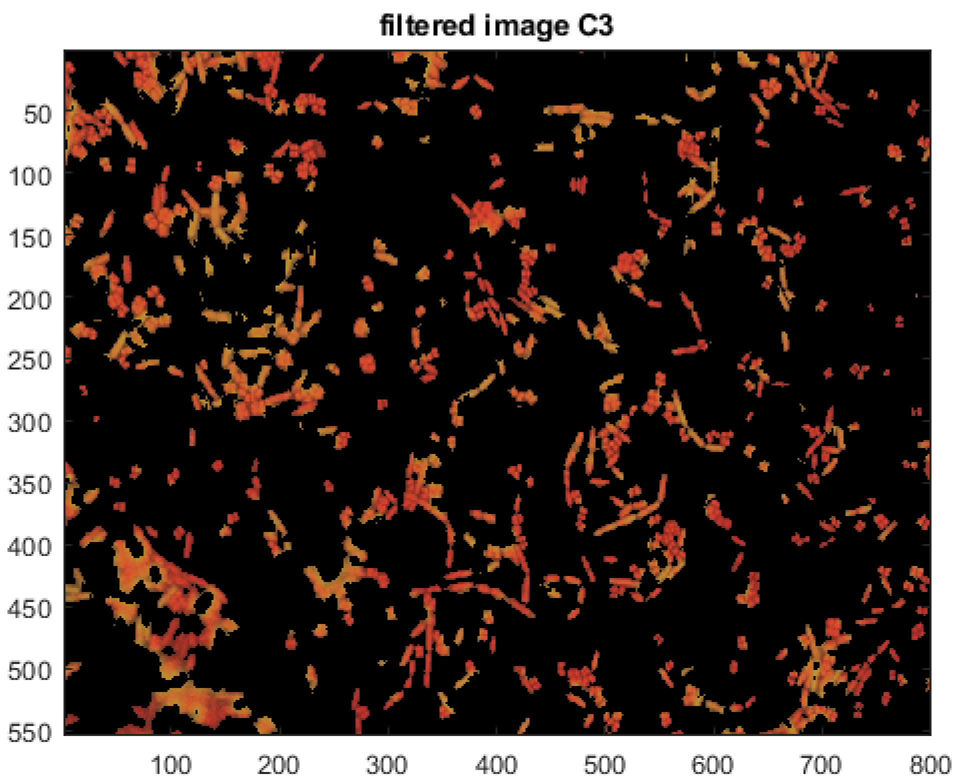
Part 3

```
A = double(imread('Bacteria_image01.tif'));

red = A(:,:,1);
green = A(:,:,2);
blue = A(:,:,3);

%because we're essentially looking for 3 classes (red, y/g and
 background)
%k = 3

X = [red(:), green(:), blue(:)];


[class,cent] = kmeans(X,3,'Replicates',50);

figure;

for i = 1:3
    cl = i == class;
    plot3(red(cl),green(cl),blue(cl),'.','MarkerSize',12)
    hold on
end
grid on
legend('C1','C2','C3')
title('K-means RGB Intensities of Bacteria')
xlabel('red intensities')
ylabel('green intensities')
zlabel('blue intensities')

E = NaN*ones(size(X));
E(1==class,:) = X(1==class,:);
E = uint8(reshape(E,size(A)));
figure;
image(E)
title('filtered image C1')


F = NaN*ones(size(X));
F(2==class,:) = X(2==class,:);
F = uint8(reshape(F,size(A)));
figure;
image(F)
title('filtered image C2')

G = NaN*ones(size(X));
G(3==class,:) = X(3==class,:);
G = uint8(reshape(G,size(A)));
figure;
image(G)
title('filtered image C3')
```


K-means RGB Intensities of Bacteria
C1
C2
C3
blue intensities
green intensities
red intensities
0
20
40
60
80
100
0
50
100
150
200
0
50
100
150
200


filtered image C1
50
100
150
200
250
300
350
400
450
500
550
100
200
300
400
500
600
700
800


filtered image C2
50
100
150
200
250
300
350
400
450
500
550
100
200
300
400
500
600
700
800


filtered image C3
50
100
150
200
250
300
350
400
450
500
550
100
200
300
400
500
600
700
800

Published with MATLAB® R2018b

HW3_P3_Jha_Vibhav

April 27, 2021

0.1 HW3 Problem 3

0.2 Name: Vibhav Jha

0.2.1 Imports

[149]:
```
import struct
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
import random
```

0.2.2 1. Loading the data

[3]:
```
# training images
with open('train-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    train_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    train_data = train_data.reshape((size, nrows, ncols))
# training labels
with open('train-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    train_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))

# test images
with open('t10k-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    test_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    test_data = test_data.reshape((size, nrows, ncols))

# test labels
with open('t10k-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    test_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
```

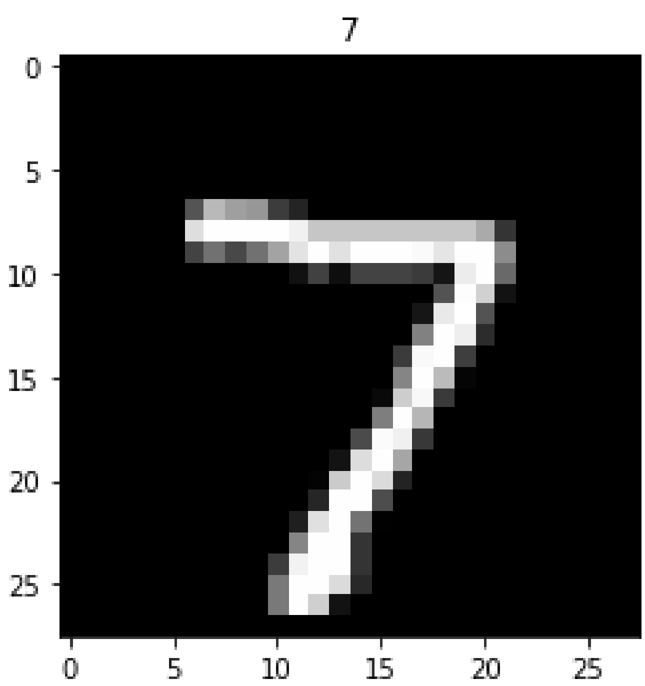
[5]:
```
print('Train Data Shape', np.shape(train_data))

print('Train Label Shape', np.shape(train_labels))

print('Test Data Shape', np.shape(test_data))

print('Test Label Shape', np.shape(test_labels))
```

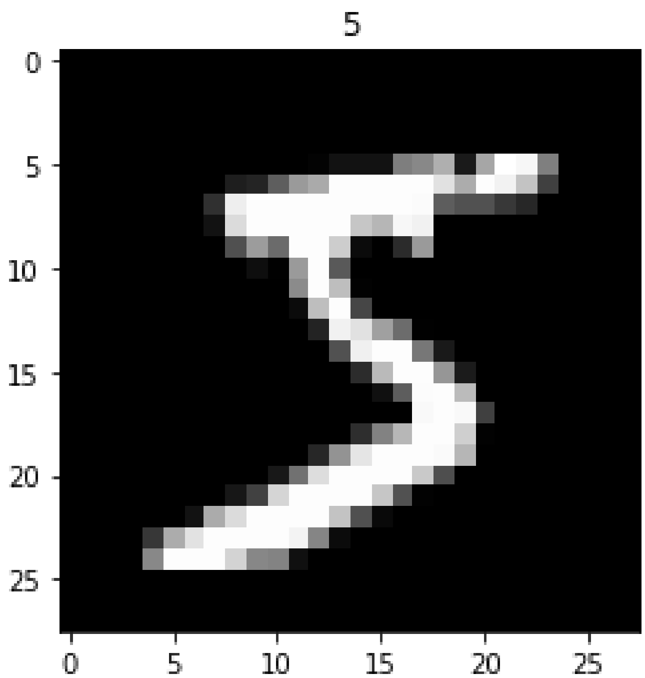
```
Train Data Shape (60000, 28, 28)
Train Label Shape (60000,)
Test Data Shape (10000, 28, 28)
Test Label Shape (10000,)
```

c. Image plots

[10]:
```
plt.figure(1)
plt.imshow(test_data[0], cmap='gray')
plt.title(test_labels[0])

plt.figure(2)
plt.imshow(train_data[0], cmap='gray')
plt.title(train_labels[0])

print('The labels and images match.')
```

```
The labels and images match.
```



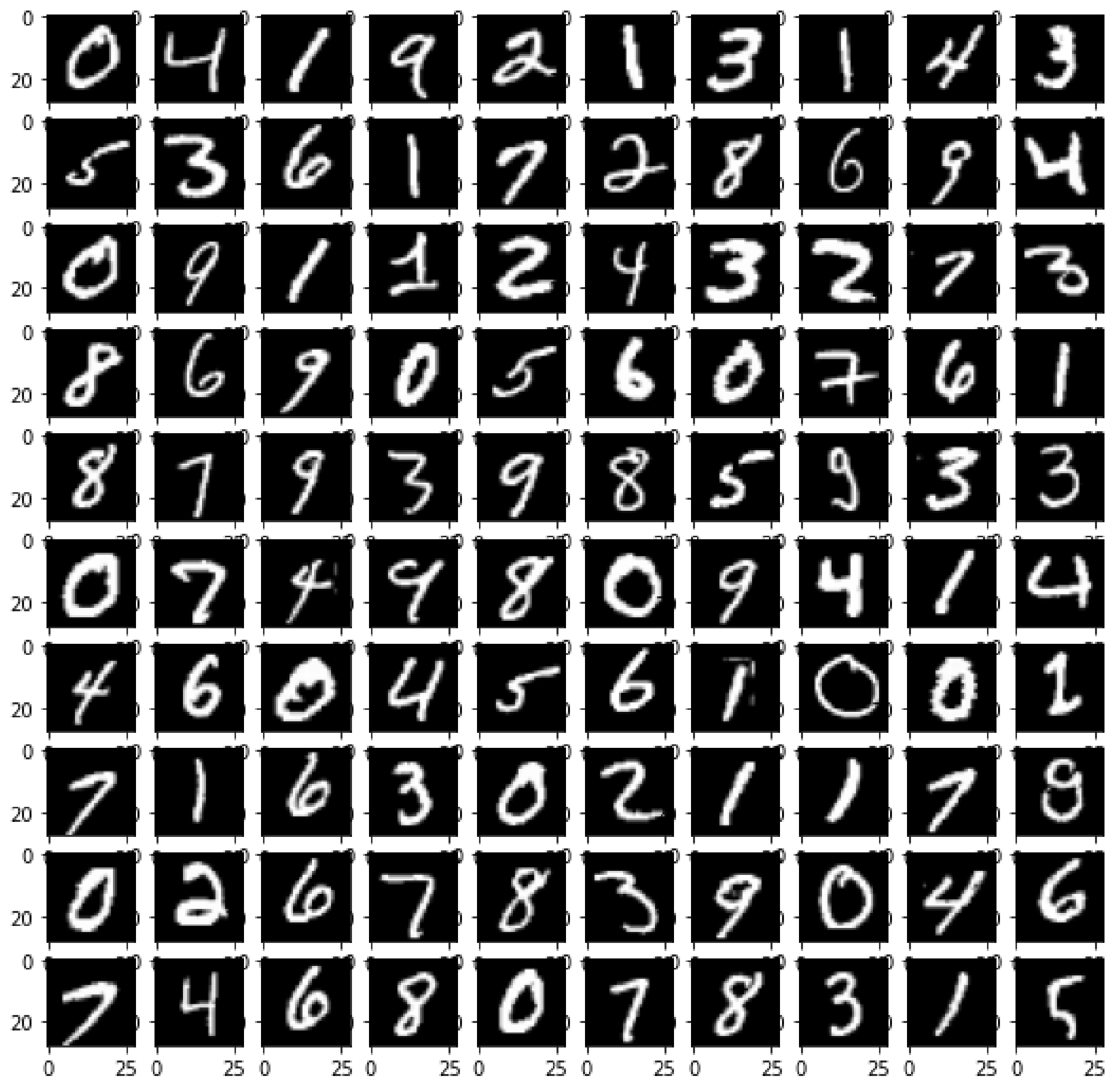


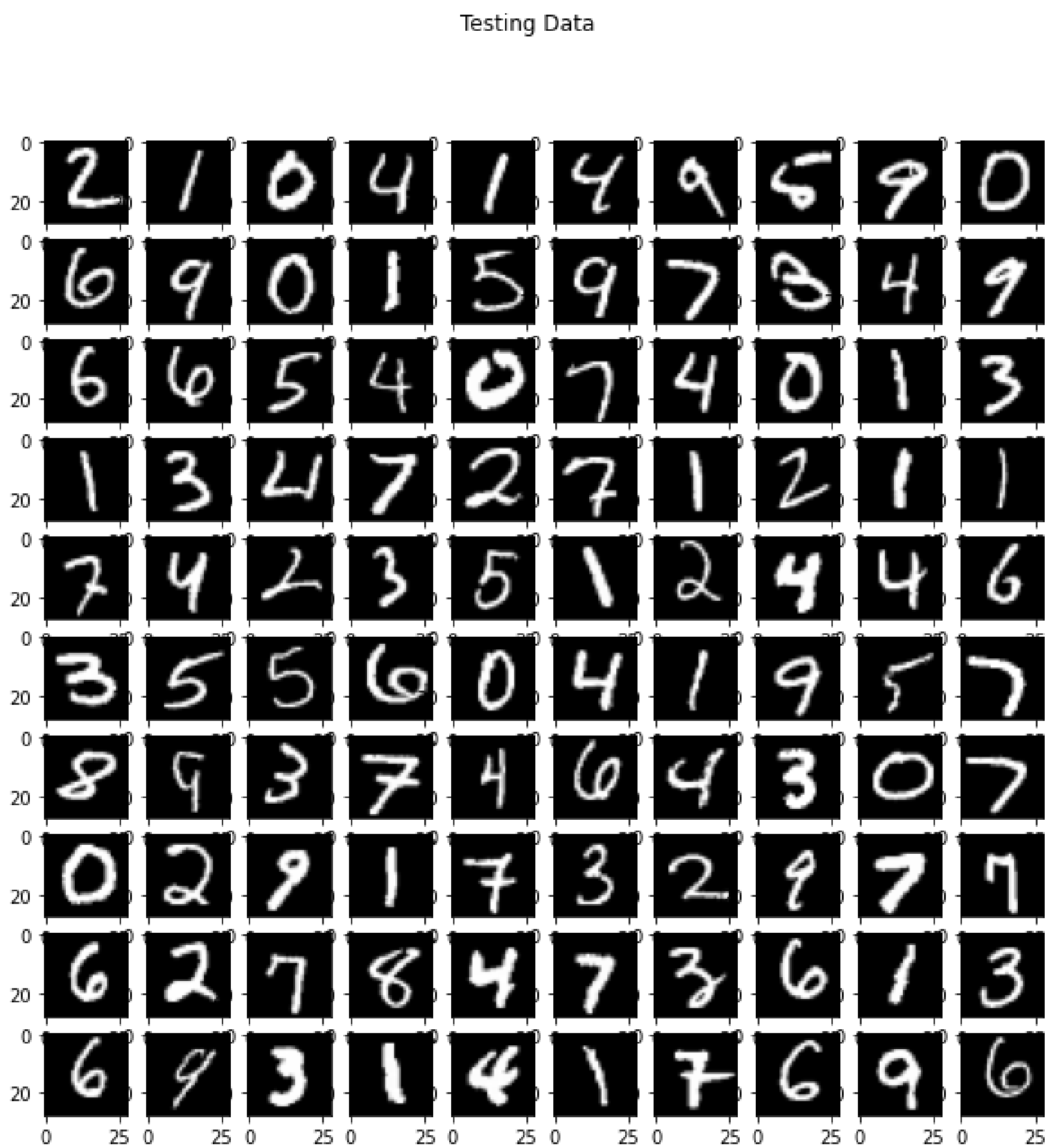
d. Image plot (10x10 grid)

[15]:
```
plt.subplots(figsize=(10,10))
for c in range(10):
    for r in range(10):
        a = r*10 + c + 1
        plt.subplot(10,10,a)
        plt.imshow(train_data[a], cmap = 'gray')
plt.suptitle('Training Data')

plt.subplots(figsize=(10,10))
for c in range(10):
    for r in range(10):
        a = r*10 + c + 1
        plt.subplot(10,10,a)
        plt.imshow(test_data[a], cmap = 'gray')
plt.suptitle('Testing Data')
```

[15]:
```
Text(0.5, 0.98, 'Testing Data')
```

Training Data





e. Digit frequency

[29]:
```
trainlab100 = train_labels[:100]
testlab100 = test_labels[:100]
#a = np.where(trainlab100==0)
#print(a)
#np.size(a)

print('Occurences of 0 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==0)))
```

```
print('Occurences of 1 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==1)))
print('Occurences of 2 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==2)))
print('Occurences of 3 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==3)))
print('Occurences of 4 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==4)))
print('Occurences of 5 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==5)))
print('Occurences of 6 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==6)))
print('Occurences of 7 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==7)))
print('Occurences of 8 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==8)))
print('Occurences of 9 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==9)))
print('')
print('Occurences of 0 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==0)))
print('Occurences of 1 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==1)))
print('Occurences of 2 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==2)))
print('Occurences of 3 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==3)))
print('Occurences of 4 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==4)))
print('Occurences of 5 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==5)))
print('Occurences of 6 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==6)))
print('Occurences of 7 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==7)))
print('Occurences of 8 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==8)))
print('Occurences of 9 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==9)))
```

```
Occurences of 0 in the first 100 train labels:  13
Occurences of 1 in the first 100 train labels:  14
Occurences of 2 in the first 100 train labels:  6
Occurences of 3 in the first 100 train labels:  11
Occurences of 4 in the first 100 train labels:  11
Occurences of 5 in the first 100 train labels:  5
```

```
Occurences of 6 in the first 100 train labels:  11
Occurences of 7 in the first 100 train labels:  10
Occurences of 8 in the first 100 train labels:  8
Occurences of 9 in the first 100 train labels:  11

Occurences of 0 in the first 100 test labels:  8
Occurences of 1 in the first 100 test labels:  14
Occurences of 2 in the first 100 test labels:  8
Occurences of 3 in the first 100 test labels:  11
Occurences of 4 in the first 100 test labels:  14
Occurences of 5 in the first 100 test labels:  7
Occurences of 6 in the first 100 test labels:  10
Occurences of 7 in the first 100 test labels:  15
Occurences of 8 in the first 100 test labels:  2
Occurences of 9 in the first 100 test labels:  11
```

0.2.3 2. Data prepartion

Normalization and reshaping

[296]:
```
train_labels6 = train_labels[:6000]
train_data6 = train_data[:6000]
print('Occurences of 0 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==0)))
print('Occurences of 1 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==1)))
print('Occurences of 2 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==2)))
print('Occurences of 3 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==3)))
print('Occurences of 4 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==4)))
print('Occurences of 5 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==5)))
print('Occurences of 6 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==6)))
print('Occurences of 7 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==7)))
print('Occurences of 8 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==8)))
print('Occurences of 9 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==9)))

#scaler = sklearn.preprocessing.MinMaxScaler()
#train_norm = sklearn.preprocessing.MinMaxScaler(train_data6)
#train_norm = scaler.train_data6_
#np.shape(train_norm) this didn't work for some reason, so doing it manually␣
 ↪using a for loop
```

```
#same story for preprocessing.normalize()

train_data_norm = np.zeros([0,784])
for i in train_data6: #general normalization formula
    train_data_norm = np.vstack((train_data_norm,((i - np.min(i)) / (np.max(i) -␣
 ↪np.min((i)))).flatten()))

train_data_norm = train_data_norm.T #transpose to get ndims x nsamples
print('New Train Data Shape: ',np.shape(train_data_norm))

test_data_norm = np.zeros([0,784])
for ii in test_data:
    test_data_norm = np.vstack((test_data_norm,((ii - np.min(i)) / (np.max(ii) -␣
 ↪np.min((ii)))).flatten()))

test_data_norm = test_data_norm.T
print('New Test Data Shape: ',np.shape(test_data_norm))
```

```
Occurences of 0 in the first 6000 train labels:  592
Occurences of 1 in the first 6000 train labels:  671
Occurences of 2 in the first 6000 train labels:  581
Occurences of 3 in the first 6000 train labels:  608
Occurences of 4 in the first 6000 train labels:  623
Occurences of 5 in the first 6000 train labels:  514
Occurences of 6 in the first 6000 train labels:  608
Occurences of 7 in the first 6000 train labels:  651
Occurences of 8 in the first 6000 train labels:  551
Occurences of 9 in the first 6000 train labels:  601
New Train Data Shape:  (784, 6000)
New Test Data Shape:  (784, 10000)
```

One-hot encoding of labels

[297]:
```
train_int = np.zeros([0,10])

for jj in range(6000):
    train_int = np.vstack((train_int, (np.arange(10) == train_labels[jj].
 ↪astype(int))))
train_int = train_int.T

print('1-hot encoded Train Labels: ',np.shape(train_int))
print('Train', train_labels[0], ' ', train_int[:,0])

test_int = np.zeros([0,10])
for jj in range(10000):
    test_int = np.vstack((test_int, (np.arange(10) == test_labels[jj].
 ↪astype(int))))
test_int = test_int.T
```

```
print('1-hot encoded Test Labels: ',np.shape(test_int))
print('Test ', test_labels[0], ' ', test_int[:,0])
```

```
1-hot encoded Train Labels:  (10, 6000)
Train 5   [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
1-hot encoded Test Labels:  (10, 10000)
Test  7   [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

0.2.4 3. Neural Network

Computational graph

[62]:
```
tf.reset_default_graph() #heavily adapted from Michelucci p.110-111 and recitatio
#784 as it is 28*28
learning_rate = tf.placeholder(tf.float64, shape=())
X = tf.placeholder(tf.float64, [784, None])
Y = tf.placeholder(tf.float64, [10, None])

#want 10 neurons, so 10 weights
weights = tf.Variable(tf.random_normal(shape = [10, 784], dtype=tf.float64,␣
↪seed=12345))

bias = tf.Variable(tf.zeros([10,1], tf.float64))

out = tf.sigmoid(tf.matmul(weights, X) + bias)

#from Michelucci p108 with the inclusion of the no nan from p77
cost = - tf.reduce_mean(tf.math.multiply_no_nan(Y,tf.log(out)) + tf.math.
↪multiply_no_nan((1-Y),tf.log(1-out)), axis=1 )

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()

saver = tf.train.Saver()
```

Training function

[114]:
```
def mnist_trainer(epochs, trainx, trainlabels, lrate, costf, optimizerf):
    sess = tf.Session()
    sess.run(init)

    cost_history = np.empty(shape=[10], dtype = float)
    for ee in range(epochs):
        [_, cost_] = sess.run([optimizer, cost], feed_dict = {X: trainx, Y:
↪trainlabels, learning_rate:lrate})

        cost_history = np.append(cost_history, np.mean(cost_))
```

```
        if (ee%500 == 0):
            print('Cost = ', np.mean(cost_), 'at epoch ', ee)
            save_mod = saver.save(sess,'trained_model_' + str(lrate) + '_' +␣
↪str(epochs) + '.ckpt')
    return sess, cost_history, save_mod


s1, chist1, saved1 = mnist_trainer(epochs = 10001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.05,
                                   costf = cost,
                                   optimizerf = optimizer)
s1.close()

s2, chist2, saved2 = mnist_trainer(epochs = 50001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.05,
                                   costf = cost,
                                   optimizerf = optimizer)
s2.close()

s3, chist3, saved3 = mnist_trainer(epochs = 50001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.01,
                                   costf = cost,
                                   optimizerf = optimizer)
s3.close()
```

```
Cost =  2.1182936721476553 at epoch  0
Cost =  0.27998292388249524 at epoch  500
Cost =  0.20307103433352286 at epoch  1000
Cost =  0.17104236086479957 at epoch  1500
Cost =  0.1523633175459273 at epoch  2000
Cost =  0.1397022359518514 at epoch  2500
Cost =  0.13033701063250494 at epoch  3000
Cost =  0.1230082450100857 at epoch  3500
Cost =  0.11704592620593783 at epoch  4000
Cost =  0.11205470319914505 at epoch  4500
Cost =  0.10778404062490321 at epoch  5000
Cost =  0.10406686941551509 at epoch  5500
Cost =  0.10078706993627766 at epoch  6000
Cost =  0.09786095486433724 at epoch  6500
Cost =  0.09522634659131059 at epoch  7000
Cost =  0.09283589400966233 at epoch  7500
Cost =  0.09065277318198824 at epoch  8000
```

```
Cost =  0.08864780065633096 at epoch  8500
Cost =  0.08679744633296001 at epoch  9000
Cost =  0.08508244598020351 at epoch  9500
Cost =  0.08348681929148014 at epoch  10000
Cost =  2.1182936721476553 at epoch  0
Cost =  0.27998292388249524 at epoch  500
Cost =  0.20307103433352286 at epoch  1000
Cost =  0.17104236086479957 at epoch  1500
Cost =  0.1523633175459273 at epoch  2000
Cost =  0.1397022359518514 at epoch  2500
Cost =  0.13033701063250494 at epoch  3000
Cost =  0.1230082450100857 at epoch  3500
Cost =  0.11704592620593783 at epoch  4000
Cost =  0.11205470319914505 at epoch  4500
Cost =  0.10778404062490321 at epoch  5000
Cost =  0.10406686941551509 at epoch  5500
Cost =  0.10078706993627766 at epoch  6000
Cost =  0.09786095486433724 at epoch  6500
Cost =  0.09522634659131059 at epoch  7000
Cost =  0.09283589400966233 at epoch  7500
Cost =  0.09065277318198824 at epoch  8000
Cost =  0.08864780065633096 at epoch  8500
Cost =  0.08679744633296001 at epoch  9000
Cost =  0.08508244598020351 at epoch  9500
Cost =  0.08348681929148014 at epoch  10000
Cost =  0.08199716189054078 at epoch  10500
Cost =  0.08060212276116276 at epoch  11000
Cost =  0.07929200940329773 at epoch  11500
Cost =  0.07805848371783727 at epoch  12000
Cost =  0.07689432438626831 at epoch  12500
Cost =  0.07579323908763191 at epoch  13000
Cost =  0.07474971456184852 at epoch  13500
Cost =  0.07375889565799124 at epoch  14000
Cost =  0.07281648676737945 at epoch  14500
Cost =  0.07191867072993789 at epoch  15000
Cost =  0.0710620415604673 at epoch  15500
Cost =  0.07024354826138748 at epoch  16000
Cost =  0.06946044764749522 at epoch  16500
Cost =  0.06871026457507187 at epoch  17000
Cost =  0.06799075830004488 at epoch  17500
Cost =  0.06729989393229985 at epoch  18000
Cost =  0.06663581813682132 at epoch  18500
Cost =  0.06599683837725842 at epoch  19000
Cost =  0.06538140511582786 at epoch  19500
Cost =  0.06478809648212798 at epoch  20000
Cost =  0.06421560500643417 at epoch  20500
Cost =  0.0636627260828897 at epoch  21000
Cost =  0.06312834788654638 at epoch  21500
```

```
Cost =  0.06261144251696187 at epoch  22000
Cost =  0.06211105818142225 at epoch  22500
Cost =  0.06162631226407052 at epoch  23000
Cost =  0.061156385154413365 at epoch  23500
Cost =  0.060700514730863554 at epoch  24000
Cost =  0.06025799141300995 at epoch  24500
Cost =  0.059828153710953694 at epoch  25000
Cost =  0.05941038421192426 at epoch  25500
Cost =  0.0590041059540067 at epoch  26000
Cost =  0.05860877914459871 at epoch  26500
Cost =  0.05822389818747583 at epoch  27000
Cost =  0.0578489889873679 at epoch  27500
Cost =  0.05748360650492249 at epoch  28000
Cost =  0.05712733253805903 at epoch  28500
Cost =  0.05677977370816147 at epoch  29000
Cost =  0.05644055963147156 at epoch  29500
Cost =  0.05610934125760294 at epoch  30000
Cost =  0.05578578935841618 at epoch  30500
Cost =  0.05546959315170966 at epoch  31000
Cost =  0.0551604590453675 at epoch  31500
Cost =  0.054858109488814685 at epoch  32000
Cost =  0.05456228191987569 at epoch  32500
Cost =  0.054272727796402555 at epoch  33000
Cost =  0.05398921170329844 at epoch  33500
Cost =  0.053711510526779846 at epoch  34000
Cost =  0.05343941268884652 at epoch  34500
Cost =  0.053172717435953 at epoch  35000
Cost =  0.05291123417676127 at epoch  35500
Cost =  0.052654781864613545 at epoch  36000
Cost =  0.05240318842099132 at epoch  36500
Cost =  0.05215629019674043 at epoch  37000
Cost =  0.05191393146824835 at epoch  37500
Cost =  0.0516759639660939 at epoch  38000
Cost =  0.05144224643394315 at epoch  38500
Cost =  0.05121264421568078 at epoch  39000
Cost =  0.05098702886893545 at epoch  39500
Cost =  0.050765277803300216 at epoch  40000
Cost =  0.050547273941673486 at epoch  40500
Cost =  0.050332905403250736 at epoch  41000
Cost =  0.050122065206799 at epoch  41500
Cost =  0.04991465099292798 at epoch  42000
Cost =  0.04971056476415907 at epoch  42500
Cost =  0.04950971264166353 at epoch  43000
Cost =  0.049312004637616216 at epoch  43500
Cost =  0.049117354442171454 at epoch  44000
Cost =  0.04892567922413048 at epoch  44500
Cost =  0.04873689944442455 at epoch  45000
Cost =  0.04855093868158693 at epoch  45500
```

```
Cost =  0.04836772346843875 at epoch  46000
Cost =  0.04818718313925023 at epoch  46500
Cost =  0.04800924968668656 at epoch  47000
Cost =  0.04783385762788032 at epoch  47500
Cost =  0.04766094387901065 at epoch  48000
Cost =  0.047490447637803294 at epoch  48500
Cost =  0.047322310273397776 at epoch  49000
Cost =  0.04715647522306051 at epoch  49500
Cost =  0.046992887895254555 at epoch  50000
Cost =  2.1182936721476553 at epoch  0
Cost =  0.6425182246590546 at epoch  500
Cost =  0.45780883553476787 at epoch  1000
Cost =  0.36604690533082984 at epoch  1500
Cost =  0.31374149726180545 at epoch  2000
Cost =  0.2800565789265448 at epoch  2500
Cost =  0.25629042694719756 at epoch  3000
Cost =  0.23838421930333062 at epoch  3500
Cost =  0.22425527033156967 at epoch  4000
Cost =  0.21272793082639768 at epoch  4500
Cost =  0.20308520646571906 at epoch  5000
Cost =  0.19486334613296252 at epoch  5500
Cost =  0.18774634309607824 at epoch  6000
Cost =  0.18150864343165932 at epoch  6500
Cost =  0.17598343456684834 at epoch  7000
Cost =  0.17104432195090086 at epoch  7500
Cost =  0.16659378014092457 at epoch  8000
Cost =  0.16255537259172503 at epoch  8500
Cost =  0.15886831611193955 at epoch  9000
Cost =  0.1554835999313468 at epoch  9500
Cost =  0.15236120382329682 at epoch  10000
Cost =  0.14946809207953493 at epoch  10500
Cost =  0.14677672892021656 at epoch  11000
Cost =  0.14426395517835408 at epoch  11500
Cost =  0.14191013689488133 at epoch  12000
Cost =  0.13969851811094794 at epoch  12500
Cost =  0.13761471801394515 at epoch  13000
Cost =  0.13564632861065215 at epoch  13500
Cost =  0.13378258677373023 at epoch  14000
Cost =  0.1320141056052944 at epoch  14500
Cost =  0.13033265475533948 at epoch  15000
Cost =  0.12873098104956815 at epoch  15500
Cost =  0.12720266183475287 at epoch  16000
Cost =  0.12574198457621596 at epoch  16500
Cost =  0.12434384743102653 at epoch  17000
Cost =  0.1230036766165854 at epoch  17500
Cost =  0.12171735730585889 at epoch  18000
Cost =  0.12048117549374653 at epoch  18500
Cost =  0.11929176882201169 at epoch  19000
```

```
Cost =  0.11814608476220453 at epoch  19500
Cost =  0.1170413448721461 at epoch  20000
Cost =  0.11597501408813278 at epoch  20500
Cost =  0.11494477421016265 at epoch  21000
Cost =  0.11394850089358974 at epoch  21500
Cost =  0.11298424358629469 at epoch  22000
Cost =  0.11205020795215673 at epoch  22500
Cost =  0.11114474040421111 at epoch  23000
Cost =  0.11026631443818313 at epoch  23500
Cost =  0.10941351851199269 at epoch  24000
Cost =  0.10858504526156557 at epoch  24500
Cost =  0.10777968187959339 at epoch  25000
Cost =  0.1069963015131153 at epoch  25500
Cost =  0.1062338555591285 at epoch  26000
Cost =  0.10549136675591038 at epoch  26500
Cost =  0.10476792298223123 at epoch  27000
Cost =  0.1040626716880412 at epoch  27500
Cost =  0.10337481488923414 at epoch  28000
Cost =  0.10270360466636763 at epoch  28500
Cost =  0.10204833911327867 at epoch  29000
Cost =  0.10140835868677107 at epoch  29500
Cost =  0.10078304291325184 at epoch  30000
Cost =  0.10017180741252871 at epoch  30500
Cost =  0.09957410120307203 at epoch  31000
Cost =  0.09898940425689177 at epoch  31500
Cost =  0.09841722527581259 at epoch  32000
Cost =  0.09785709966429947 at epoch  32500
Cost =  0.09730858767708281 at epoch  33000
Cost =  0.0967712727226265 at epoch  33500
Cost =  0.09624475980595963 at epoch  34000
Cost =  0.09572867409657199 at epoch  34500
Cost =  0.09522265960895707 at epoch  35000
Cost =  0.09472637798499325 at epoch  35500
Cost =  0.0942395073687324 at epoch  36000
Cost =  0.09376174136531926 at epoch  36500
Cost =  0.09329278807674635 at epoch  37000
Cost =  0.09283236920797565 at epoch  37500
Cost =  0.09238021923765997 at epoch  38000
Cost =  0.09193608464828511 at epoch  38500
Cost =  0.09149972321106839 at epoch  39000
Cost =  0.09107090332138318 at epoch  39500
Cost =  0.09064940338085321 at epoch  40000
Cost =  0.09023501122259872 at epoch  40500
Cost =  0.08982752357639552 at epoch  41000
Cost =  0.08942674557077243 at epoch  41500
Cost =  0.0890324902692976 at epoch  42000
Cost =  0.08864457823850709 at epoch  42500
Cost =  0.08826283714511937 at epoch  43000
```

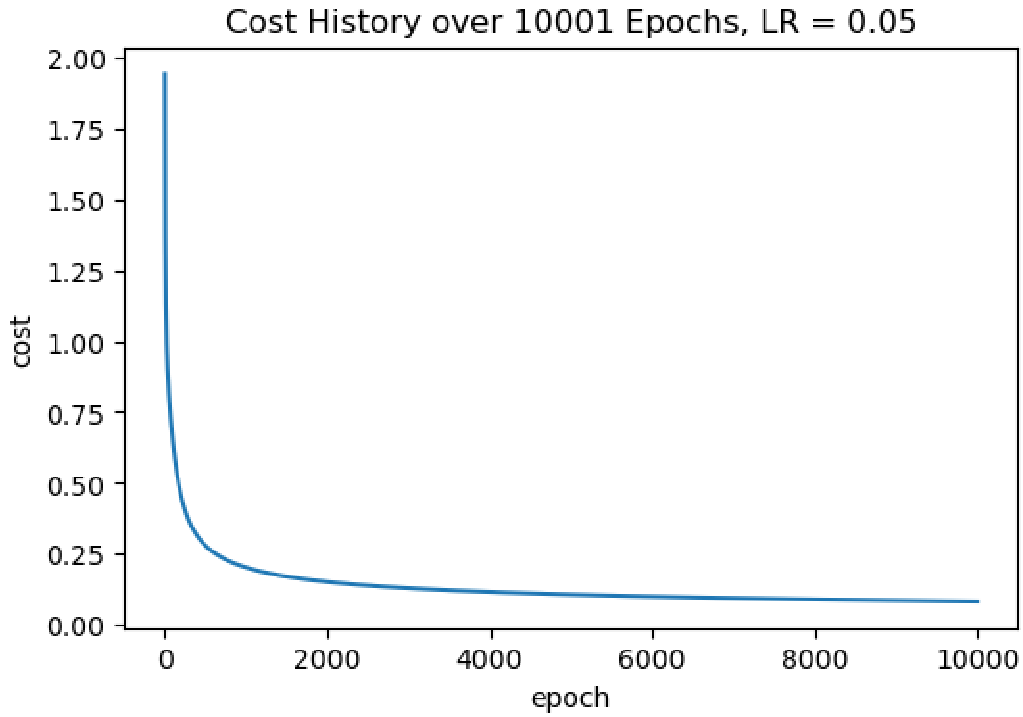
```
Cost =  0.08788710138034621 at epoch  43500
Cost =  0.08751721170926555 at epoch  44000
Cost =  0.08715301494336802 at epoch  44500
Cost =  0.08679436363452313 at epoch  45000
Cost =  0.08644111578873956 at epoch  45500
Cost =  0.08609313459820822 at epoch  46000
Cost =  0.0857502881902357 at epoch  46500
Cost =  0.08541244939177844 at epoch  47000
Cost =  0.08507949550838974 at epoch  47500
Cost =  0.08475130811648801 at epoch  48000
Cost =  0.0844277728679423 at epoch  48500
Cost =  0.08410877930605673 at epoch  49000
Cost =  0.0837942206921122 at epoch  49500
Cost =  0.08348399384169977 at epoch  50000
```

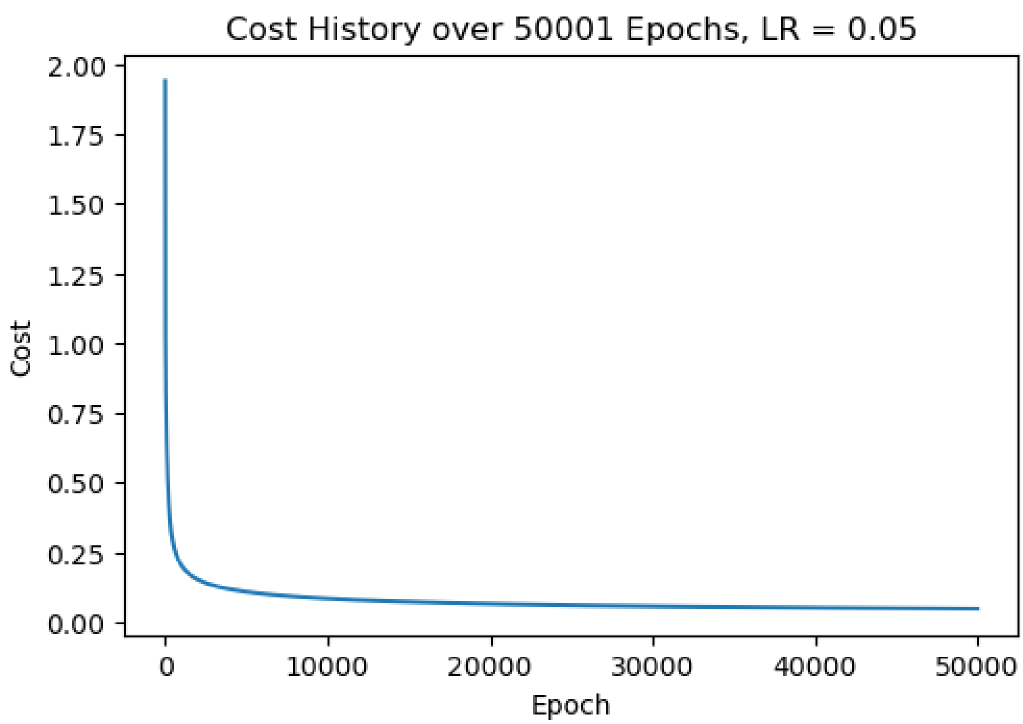
0.2.5 4. Training and testing

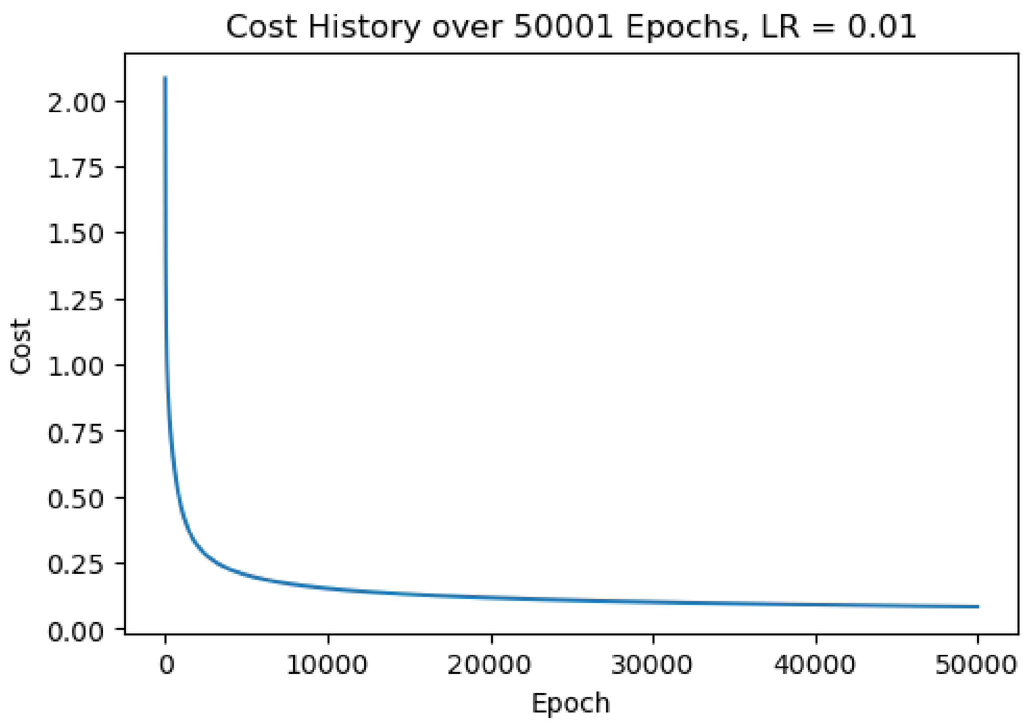
a. Cost history

[295]:
```
plt.figure()

#print(np.shape(chist1))
plt.plot(chist1[11:])
plt.title('Cost History over 10001 Epochs, LR = 0.05')
plt.xlabel('epoch')
plt.ylabel('cost')


plt.figure()
plt.plot(chist2[11:])
plt.title('Cost History over 50001 Epochs, LR = 0.05')
plt.xlabel('Epoch')
plt.ylabel('Cost')

plt.figure()
plt.plot(chist3[11:])
plt.title('Cost History over 50001 Epochs, LR = 0.01')
plt.xlabel('Epoch')
plt.ylabel('Cost')
```

[295]:
```
Text(0, 0.5, 'Cost')
```


Cost History over 10001 Epochs, LR = 0.05
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
cost
0
2000
4000
6000
8000
10000
epoch


Cost History over 50001 Epochs, LR = 0.05
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cost
0
10000
20000
30000
40000
50000
Epoch



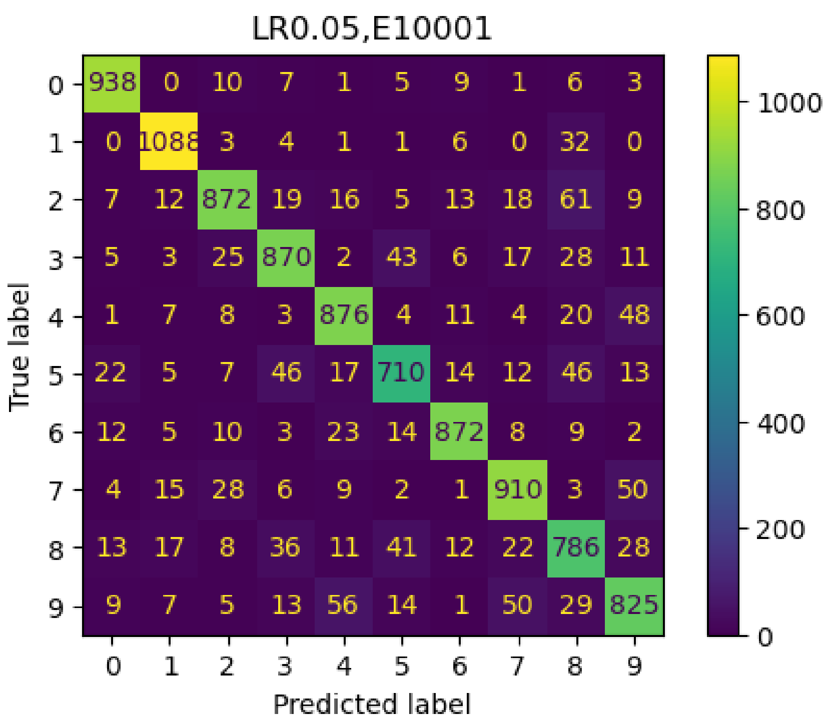
b. Confusion matrix

[142]:
```
plt.rcParams['figure.dpi'] = 100
sess = tf.Session()
saver.restore(sess, saved1)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm1 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()

ConfusionMatrixDisplay(confusion_matrix = cm1).plot()
plt.title('LR0.05,E10001')


sess.close()


sess = tf.Session()
saver.restore(sess, saved2)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm2 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()
```

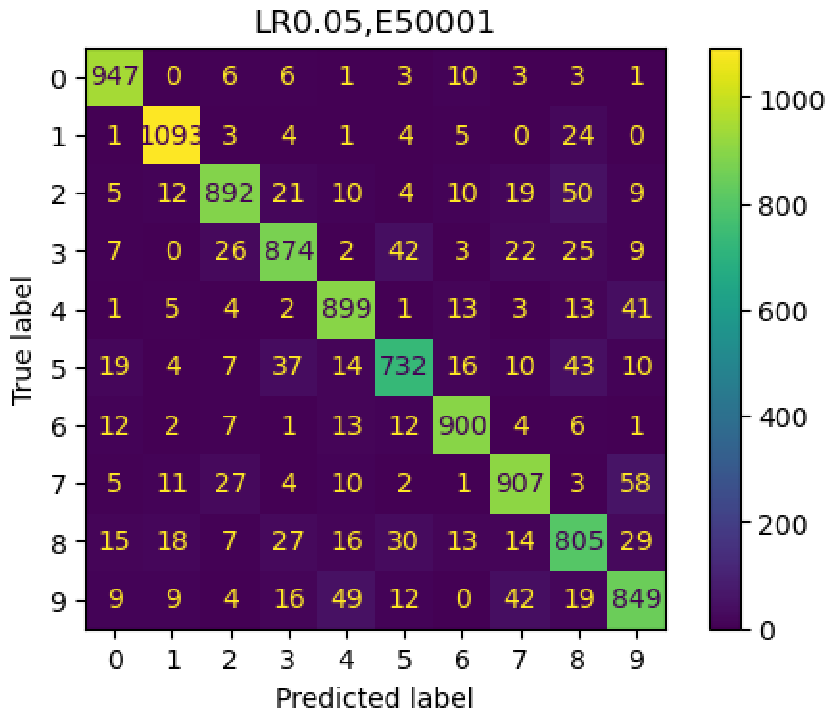
```
ConfusionMatrixDisplay(confusion_matrix = cm2).plot()
plt.title('LR0.05,E50001')
sess.close()

sess = tf.Session()
saver.restore(sess, saved3)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm3 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()

ConfusionMatrixDisplay(confusion_matrix = cm3).plot()
plt.title('LR0.01')
sess.close()
```

```
INFO:tensorflow:Restoring parameters from trained_model_0.05_10001.ckpt
INFO:tensorflow:Restoring parameters from trained_model_0.05_50001.ckpt
INFO:tensorflow:Restoring parameters from trained_model_0.01_50001.ckpt
```

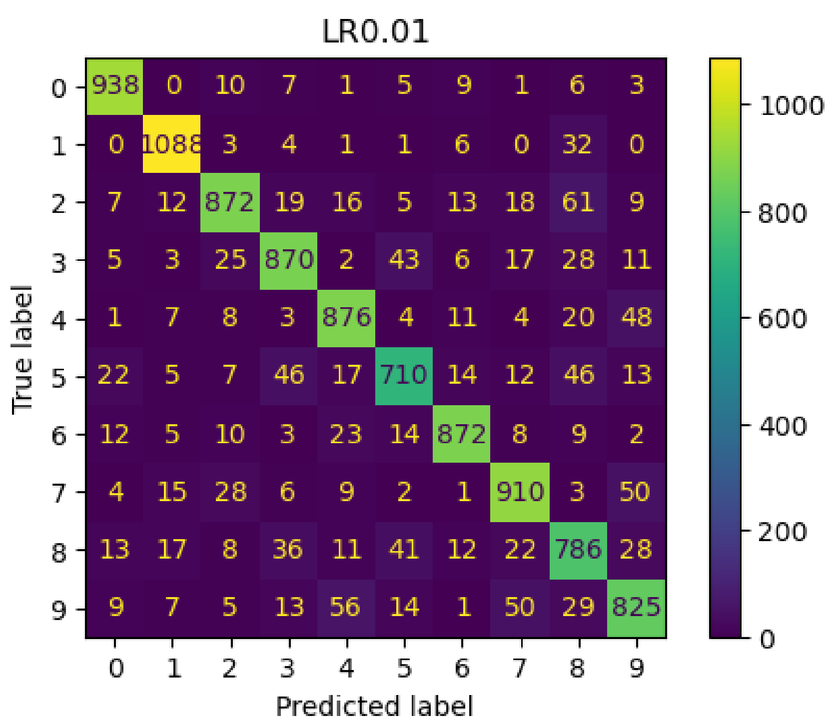
```
<Figure size 600x400 with 0 Axes>
```




```
<Figure size 600x400 with 0 Axes>
```

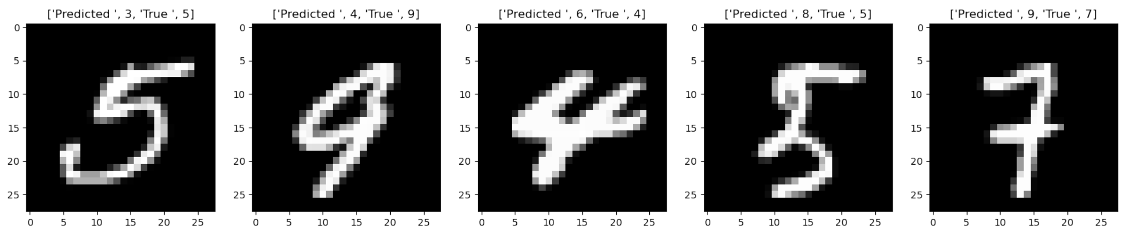


<Figure size 600x400 with 0 Axes>



c. Common misclassifications

[293]:
```
cases = [219, 7580 , 6759, 3776, 307]
plt.figure(figsize = (20,20))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(test_data[cases[i]], cmap = 'gray')
    plt.title(['Predicted ', ytestout[cases[i]], 'True ', test_labels[cases[i]]])
```



[294]:
```
print('9 and 4 can be easily misclassfied as the 9 is not always connected at␣
 ↪the top, thus looking like a 4.')
print('5 and 3 are written similarly, however if the upper stem on the five is␣
 ↪close to the rest of the number, it can begin to look a little like a␣
 ↪compacted 3.')
print('6 and 4 are most likely for a similar reason as to nine, however if we␣
 ↪trace a 6 the bottom part, if done crudely, can look like the line going␣
 ↪through a 4.')
print('8 and 5: a very compacted 5 nearly can resemble two circles on top of␣
 ↪each other, thus becoming an 8.')
print('9 and 7, if 7 is written with a horizontal line through it and possible␣
 ↪and extra vertical line from the top, it resembles a 9.')
print('Ultimately, handwriting styles are unique to a person, and depending on␣
 ↪how one chooses to write many numbers can have very similar features to␣
 ↪others, in this case we see')
print('5 and 3, 9 and 4, 8 and 5, 9 and 7')
```

```
9 and 4 can be easily misclassfied as the 9 is not always connected at the top,
thus looking like a 4.
5 and 3 are written similarly, however if the upper stem on the five is close to
the rest of the number, it can begin to look a little like a compacted 3.
6 and 4 are most likely for a similar reason as to nine, however if we trace a 6
the bottom part, if done crudely, can look like the line going through a 4.
8 and 5: a very compacted 5 nearly can resemble two circles on top of each
other, thus becoming an 8.
9 and 7, if 7 is written with a horizontal line through it and possible and
```

extra vertical line from the top, it resembles a 9.
Ultimately, handwriting styles are unique to a person, and depending on how one chooses to write many numbers can have very similar features to others, in this case we see
5 and 3, 9 and 4, 8 and 5, 9 and 7

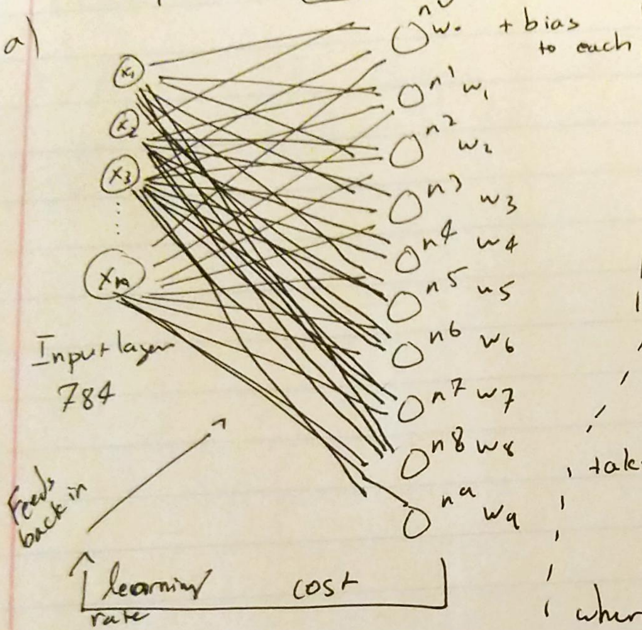
[ ]:
```

```

[ ]:
```

```

Problem 3

784 features because
shape = 1000, 28, 28 → collapse

a)



Only 1 layer of neurons, since 1v. rest
→ so its actually the output layer
using sigmoid neurons
activation fxn → $f(z) = \frac{1}{1+e^{-z}}$

cost → $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{i} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}))$

each neuron is connected to all inputs

take neuron 0 for example

$x_1 w_0 + x_2 w_0 + \ldots + x_{784} w_0$

where the w_0 corresponds to n_0, we also add the bias term

after ~~the first~~ each pass we go back and use gradient descent to optimize

b) Unless the cost was minimized, would make it as NaN and attempt to reclassify by ~~adjusting~~ the adjusted parameters.