GEOG5991M Assessment# 2 - The Black Death

Introduction

The Black Death refers to the Bubonic plague outbreak in London in 1665. It is commonly believed that the plague was caused by fleas from rats infecting people. For this project, raster maps in the form of files with digital texts were provided for the average population densities per 100mx100m for the 16 London parishes and the average numbers of rats caught by rat catchers who were assigned to 8 areas. The implementation of this project involves reading in these two files and plotting them as maps and producing a map to show the average weekly deaths which is calculated using the equation d = (0.8 x r) x (1.3 xp). The data for the average weekly deaths is written to file and is also used to calculate the total weekly deaths for London.

In [1]:

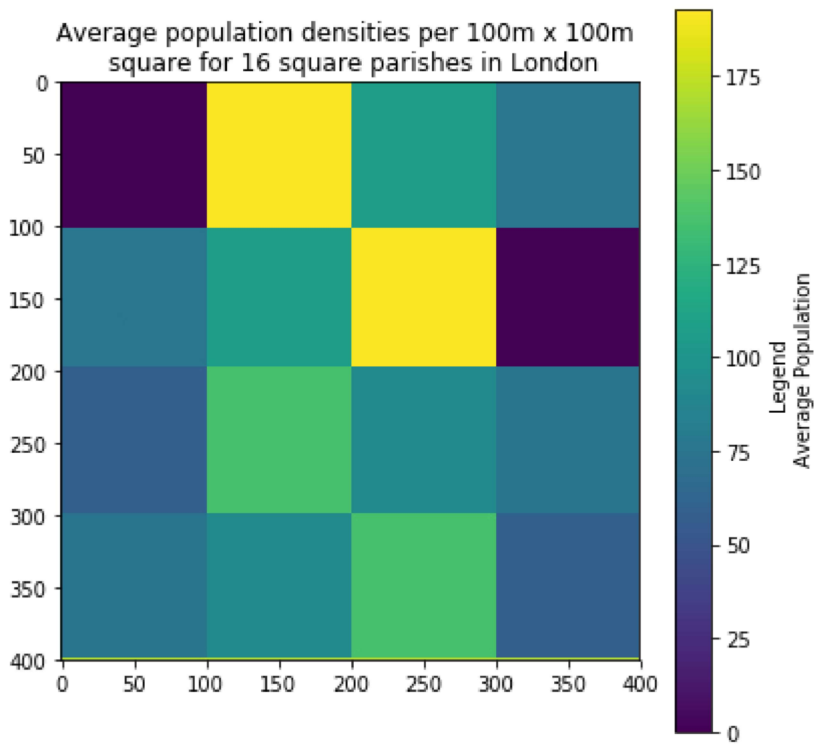
```
#import modules that will be used in the program
import matplotlib.pyplot as plt
import csv
import numpy as np

%matplotlib inline
from ipywidgets import interactive
```

In [2]:

```
#The three lists that will be used to hold data for population, rat and average deaths
population_map = []
rat_map = []
average_death_map = []
```

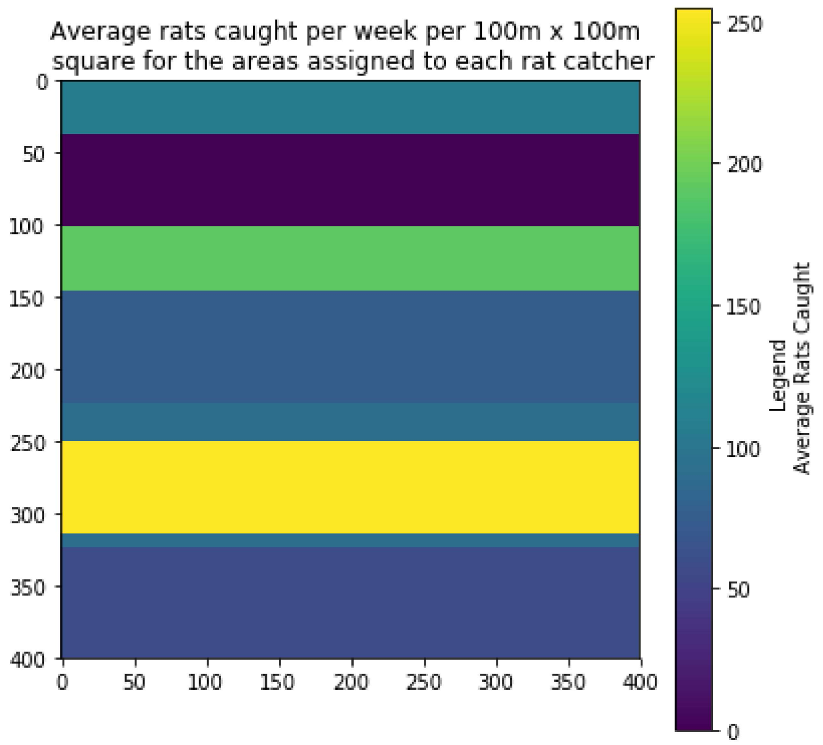
In [3]:

```
#open the file death.parishes which has data for the raster map of average population d
ensities per 100 x 100 m square averaged
#at the parish area scale.
#The data is read into a list. The list is then appended to the population_map list to
 create a 2d list or grid.
#Use features from matplotlib to plot a map of average population

try:
    f1 = open('death.parishes', newline='')
except IOError:
    print ("Could not read file:", 'death.parishes')
    exit()

f1 = open('death.parishes', newline='')
reader = csv.reader(f1, quoting=csv.QUOTE_NONNUMERIC)
for row in reader:
    rowlist1 = []
    for value in row:
        rowlist1.append(value)
    population_map.append(rowlist1)

fig, ax = plt.subplots(1, figsize=(5, 5))
ax.set_axis_off()

ax = fig.add_axes([0, 0, 1, 1])

plt.imshow(population_map)
plt.title('Average population densities per 100m x 100m \n square for 16 square parishe
s in London')
plt.colorbar(label='Legend \n Average Population')
plt.show()
f1.close()
```



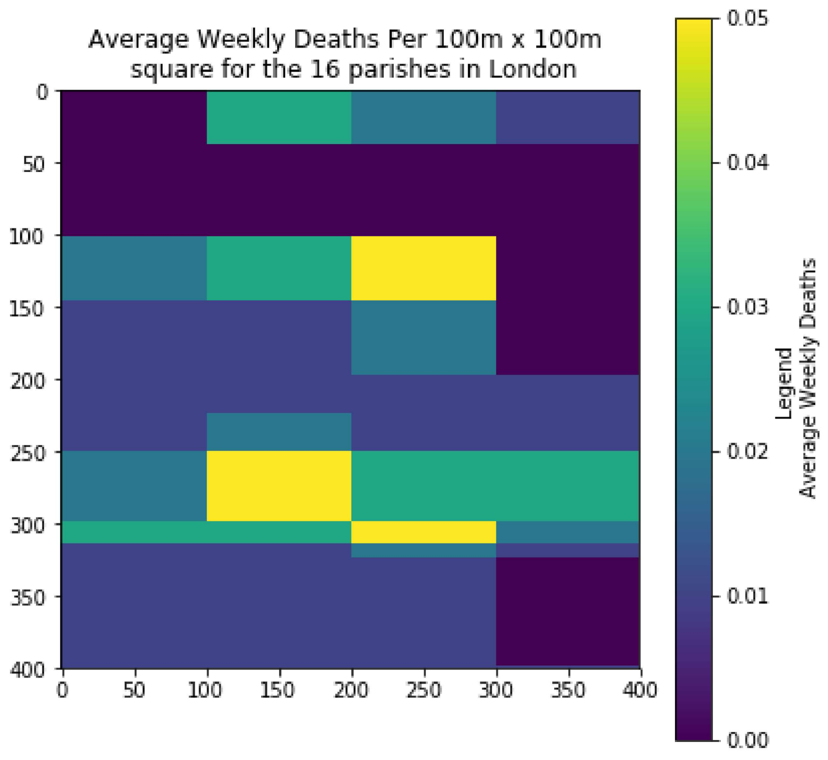
In [4]:

```
#open the file death.rats which has data for the raster map of average rats caught per
 100 x 100 m square averaged
#at the rat catchers' area scale.
#The data is read into a list. The list is then appended to the rat_map list to create
 a 2d list or grid.
#Use features from matplotlib to plot a map of average rats caught per week

#Before opening the file check that it is available
try:
    f2 = open('death.rats', newline='')
except IOError:
    print ("Could not read file:", 'death.rats')
    exit()

f2 = open('death.rats', newline='')
reader = csv.reader(f2, quoting=csv.QUOTE_NONNUMERIC)
for row in reader:
    rowlist2 = []
    for value in row:
        rowlist2.append(value)
    rat_map.append(rowlist2)

fig, ax = plt.subplots(1, figsize=(5, 5))
ax.set_axis_off()

ax = fig.add_axes([0, 0, 1, 1])
plt.imshow(rat_map)
plt.title('Average rats caught per week per 100m x 100m \n square for the areas assigne
d to each rat catcher')
plt.colorbar(label='Legend \n Average Rats Caught')
plt.show()
f2.close()
```



In [5]:

```
"""Create two numpy array from the lists for the population_map and rat_map because num
py array allows us to do element-wise
#calculations easier than using a list which would require implementing for loops"""

ave_pop_arr = np.array(population_map)
ave_rat_arr = np.array(rat_map)

#before performing any mathematical operations on the two arrays check to ensure they a
re the same shape by using assert
assert ave_pop_arr.shape == ave_rat_arr.shape

"""to use the function d = (0.8*r) * (1.3*p) is is necessary to calculate:
1. p - the average population density per 100m x 100m pixel : This is calculated as the
 number of people in each pixel divided
   by the the area of the pixel multiply by 16 because the population data was averaged
 at the parish scale
2. r - the average rats caught per week per 100m x 100m pixel : This is calculated as t
he number of rats caught in each pixel
   divided by the area of a pixel multiply by 8 because the rat cathers data was averag
ed at the Rat Cathers area scale

   convert the numpy array resulting from d = (0.8*r) * (1.3*p) to a list average_death
_map

   plot the average_death_map as a map using matplotlib"""

average_death_map = np.round(np.multiply((1.3*16*ave_pop_arr/10000),(0.8*8*ave_rat_arr/
10000)).tolist(),2)


fig, ax = plt.subplots(1, figsize=(5, 5))
ax.set_axis_off()

ax = fig.add_axes([0, 0, 1, 1])
plt.imshow(average_death_map)
plt.title('Average Weekly Deaths Per 100m x 100m \n square for the 16 parishes in Londo
n')
plt.colorbar(label='Legend \n Average Weekly Deaths')
plt.show()
```



In [6]:

```
#Write average weekly deaths to file

#Write the list contaning the data for the average_deaths to a file output.txt

f3 = open('output.txt', 'w', newline='')
writer = csv.writer(f3, quoting=csv.QUOTE_NONNUMERIC)
for row in average_death_map:
    writer.writerow(row)
f3.close()
```

In [7]:

```
"""Total Weekly Death Calculation

To calculate total deaths per week it is necessary to pull out the death count for each
 pixel:
use numpy array unique_values and count of unique_values to obtain the values in the pi
xels and the occurrences of these values
                                                                                .
Next multiply the unique values by the count or total occurrences of the values.
convert the result obtained to a list
Next use the sum() method of python list to calculate the total deaths per week and out
put this result """

unique_values, counts = np.unique(np.array(average_death_map), return_counts=True)

#print("Unique values : ", unique_values) #a check to see the unique values printed
#print("Counts : ", counts)               #a check to see values are printed for the co
unt of unique values printed

val_count = np.array(unique_values * counts).tolist()


print("Total deaths per week is : ", np.ceil(sum(val_count)))
```

```
Total deaths per week is :  2233.0
```

In [8]:

```
#Function to create interactive sliders to allow users to change parameters for the equ
ation d = (0.8 x r) x (1.3 xp)
def adjustmap(rat_param, pop_param):
    plt.figure()
    fig, ax = plt.subplots(1, figsize=(5, 5))
    ax.set_axis_off()

    ax = fig.add_axes([0, 0, 1, 1])
    plt.imshow(np.multiply((rat_param*ave_pop_arr),(pop_param*ave_rat_arr)).tolist())
    plt.show()

interactive_plot = interactive(adjustmap, rat_param=(0.0, 2.0), pop_param=(0.0, 3))
output = interactive_plot.children[1]
interactive_plot
```