Spatial Analysis and Mapping Tutorial

Cedric Elkouh, Naina Prasad*

3/5/2020

Overview

Today we will learn about using `R` to make maps with the `tmap` and `sf` packages that allow us to conduct some simple spatial analyses. This involves rendering maps in `R` and using the software as a Geographic Information System (GIS). To do this, we will work with both comma-separated values (.csv) files and GIS shapefiles. When we combine the data in the .csv and GIS shapefiles, we can learn how behavior and attributes vary over space by visualizing the relationships between and within variables. We don't cover some more advanced topics today, like spatial autocorrelation, but you are encouraged to explore those on your own time.

Setup and Data

The data we are going to use today comes from the Consumer Data Research Centre (CDRC), which provides consumer-related data from a large number of sources within the UK. We will be using data on the London Borough of Camden. Click here to get the data, and download the repository as a ZIP file if you have not already. Save the file on your Desktop and unzip it.

```
library(rgdal) # Bindings for the Geospatial Data Abstraction Library
library(rgeos) # Interface to Geometry engine - Open Source
library(tmap) # Gives us more mapping power
library(tmaptools)
library(RColorBrewer) # Gives us more colors for our maps
library(sp) # Lets us convert .csv to spatial points data frame
library(raster)
library(adehabitatHR) # For kernel density estimations

# If you get an error, use install.packages()

# Visit the link above and download the data. Then:
setwd("~/Desktop/SpatialAnalysisTutorial-master/Camden")

Census.Data <- read.csv("practical_data.csv")
Output.Areas <- readOGR(".", "Camden_oa11") # This gets us our shapefile, OA.Census
houses <- read.csv("CamdenHouseSales15.csv") # House price data
```

Take some time to explore the variables in the census data. Note that all the data in `Census.Data` are percentages and that the shapefile should contain the same number of features as the number of observations in `Census.Data`.

First we will explore mapping in `R`; then we will do some spatial analyses. There will be questions and exercises along the way.

*This tutorial is wholly based on *An Introduction to Spatial Analysis and Visualisation in R* by Guy Lansley and James Cheshire. The whole thing is available, for free, here.

Combining Our Data

First, we need to combine our shapefile with the census data to have a map that can tell us things about the space it represents.

1. *Explore the shapefile.* If we use the `plot()` function, we will get a map of our area. But you can see there is little information about the attributes of any space on the map. What information do you think must be in the shapefile to make this possible?

```
plot(Output.Areas)
```

2. *Joining data.* We now need to join `Census.Data` to the shapefile so that the census attributes can be mapped. If we use the `merge()` function, we can join the data along the unique names of each of the output areas (this is the variable `OA`). But notice that this time the column headers for our output area names are not identical, despite the fact they contain the same data. We therefore have to use `by.x` and `by.ya` so the merge function uses the correct columns to join the data.

```
OA.Census <- merge(Output.Areas, Census.Data, by.x="OA11CD", by.y="OA")
```

3. *Setting a coordinate system.* It is also important to set a coordinate system, especially if you want to map multiple different files. The `proj4string()` and `CRS()` functions allows us to set the coordinate system of a shapefile to a predefined system of our choice. Most data from the UK is projected using the British National Grid (EPSG:27700) produced by the Ordnance Survey, this includes the standard statistical geography produced by the Office for National Statistics. It turns out that our shapefile already has the correct projection system, but this step is good practice.

```
proj4string(OA.Census) <- CRS("+init=EPSG:27700")
```

Now we can finally make some interesting maps!

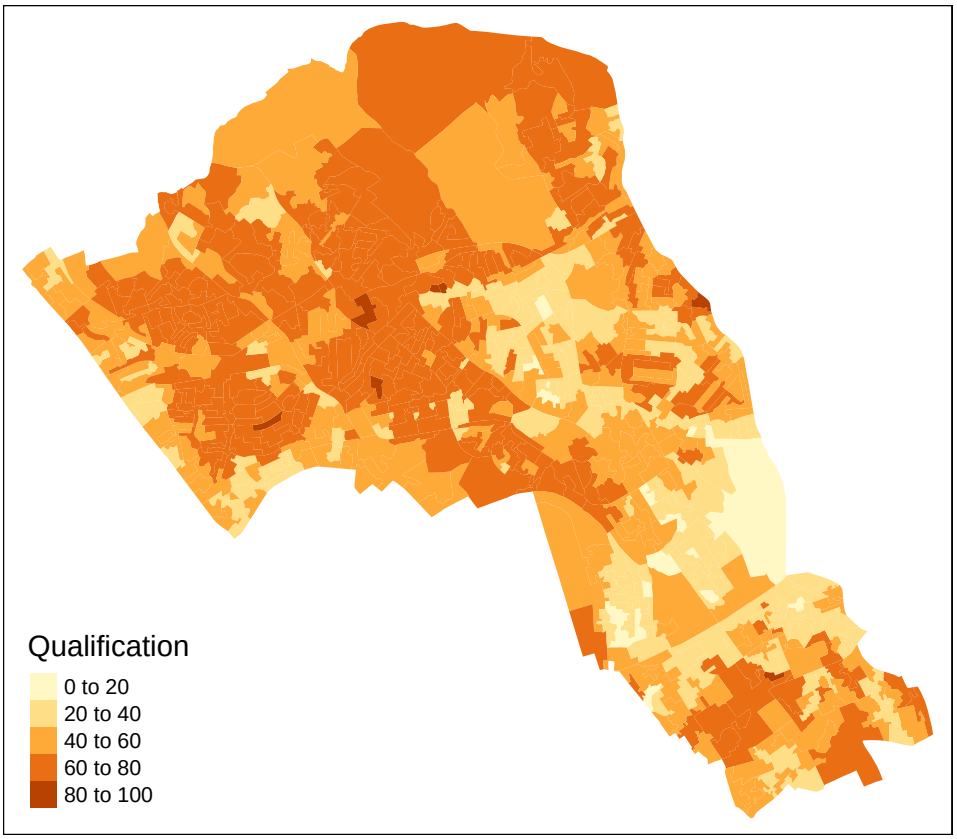
Map-Making Basics

Just like with `ggplot2`, creating more advanced maps in `tmap` involves binding together several functions that comprise different aspects of the graphic. We enter shapefiles (or R spatial data objects) followed by a command to set their symbologies. The objects are then layered in the visualisation in order, so the object entered first appears at the bottom of the graphic. We can think of two major steps:

1. Use `tm_shape()` to call a shapefile (in this case, `OA.Census`).
2. Add (`+`) parameters that further customize our map. For example, `tm_fill()` allows us to determine how the polygons are filled in the graphic.

If we use the `Qualification` variable, we can see in which areas the greatest proportion of people have attained Level 4 qualifications and above. (Level 4 qualifications refer to a Certificate of Higher Education Higher National Certificate, which is awarded by a degree-awarding body.)

```
tm_shape(OA.Census) + tm_fill("Qualification")
```



Adding More Color

If we want to use a different color scheme, we can use the color ramps predefined by the `RColorBrewer()` function. You can explore the options available to us by running `display.brewer.all()`.

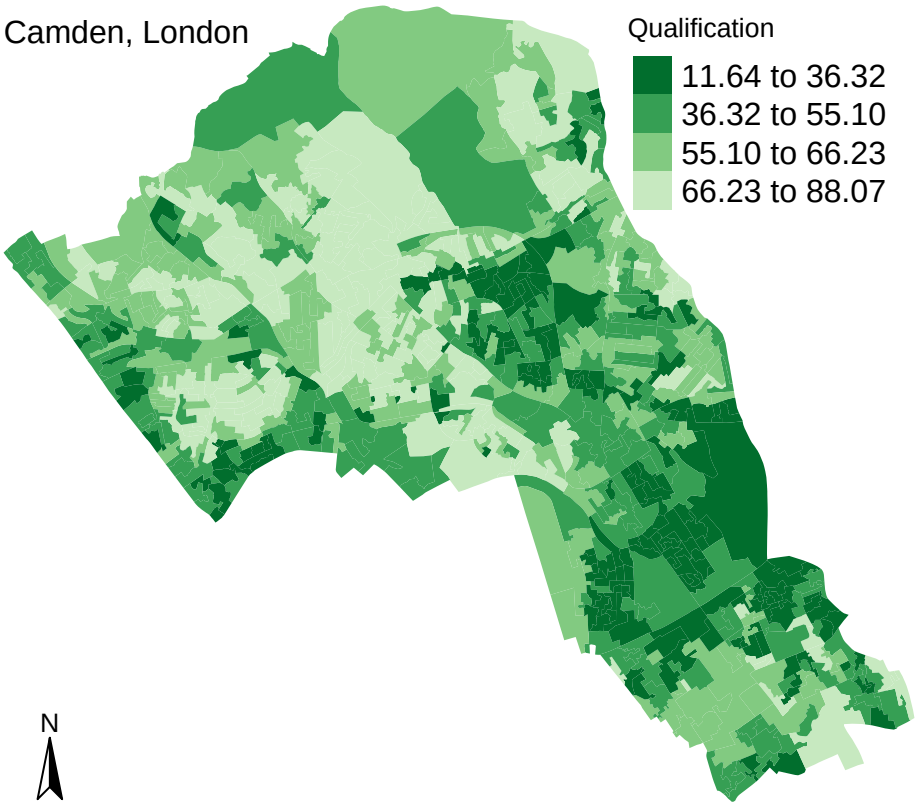
The continuous ramps you will see in the top group are all appropriate for our data. To change the color ramp, enter its name after `palette =`. If you enter a minus sign before the name of the ramp within the brackets (e.g., `-Greens`), you will invert the order of the color ramp.

We have a range of different interval options in the style parameter. Each of them will greatly impact how your data is visualised. To do this you enter `style =`, followed by one of the options below.

- `equal` - divides the range of the variable into n parts.
- `pretty` - chooses a number of breaks to fit a sequence of equally spaced 'round' values. So the keys for these intervals are always tidy and memorable.
- `quantile` - equal number of cases in each group
- `jenks` - looks for natural breaks in the data
- `Cat` - if the variable is categorical (i.e., not continuous data)

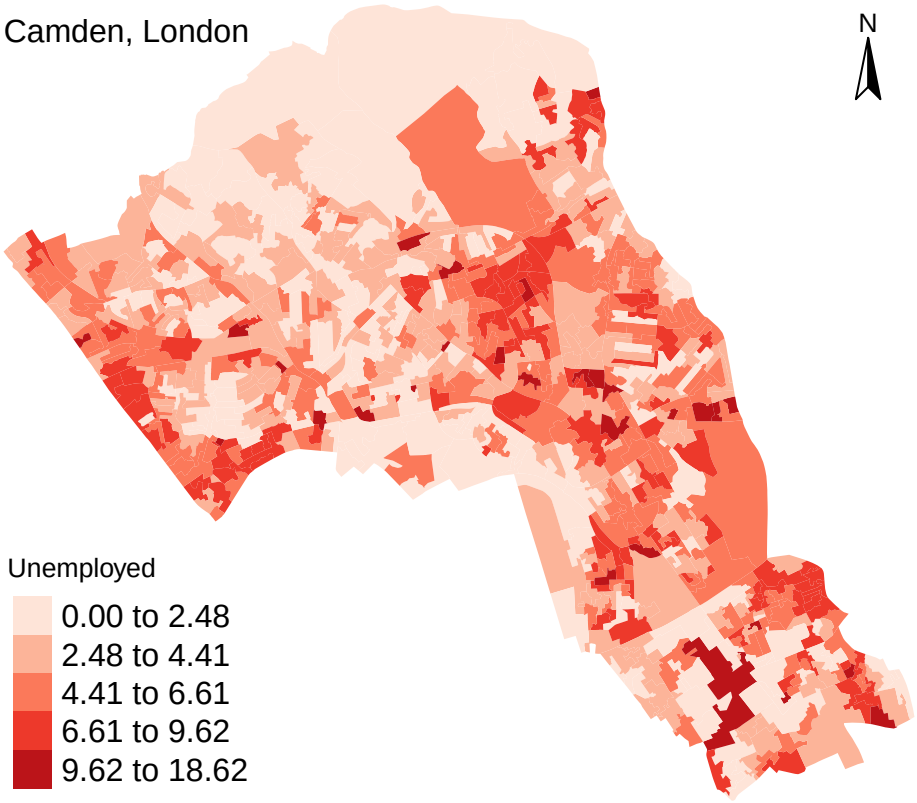
In this map, darker-green areas have a lower proportion of people who have attained Level 4 qualification and above relative to lighter-green areas. Take a look at some of the arguments in the `tm_layout()` parameter and see how changing them changes your map.

```
tm_shape(OA.Census) +
  tm_fill("Qualification", style = "quantile", n = 4, palette = "-Greens") +
  tm_compass(position = c("left", "bottom")) +
  tm_layout(title = "Camden, London", title.size = 1, legend.text.size = 1,
legend.title.size = 1, legend.position = c("right", "top"), frame = FALSE)
```



Exercise 1: Replicate the below plot

Darker-red areas indicate higher rates of unemployment. If you are having trouble figuring out the style, think about where the breaks in a histogram of the census data would fall for different styles. (You can add a histogram to the map with `legend.hist = TRUE` to make a quick comparison.)



Point Data

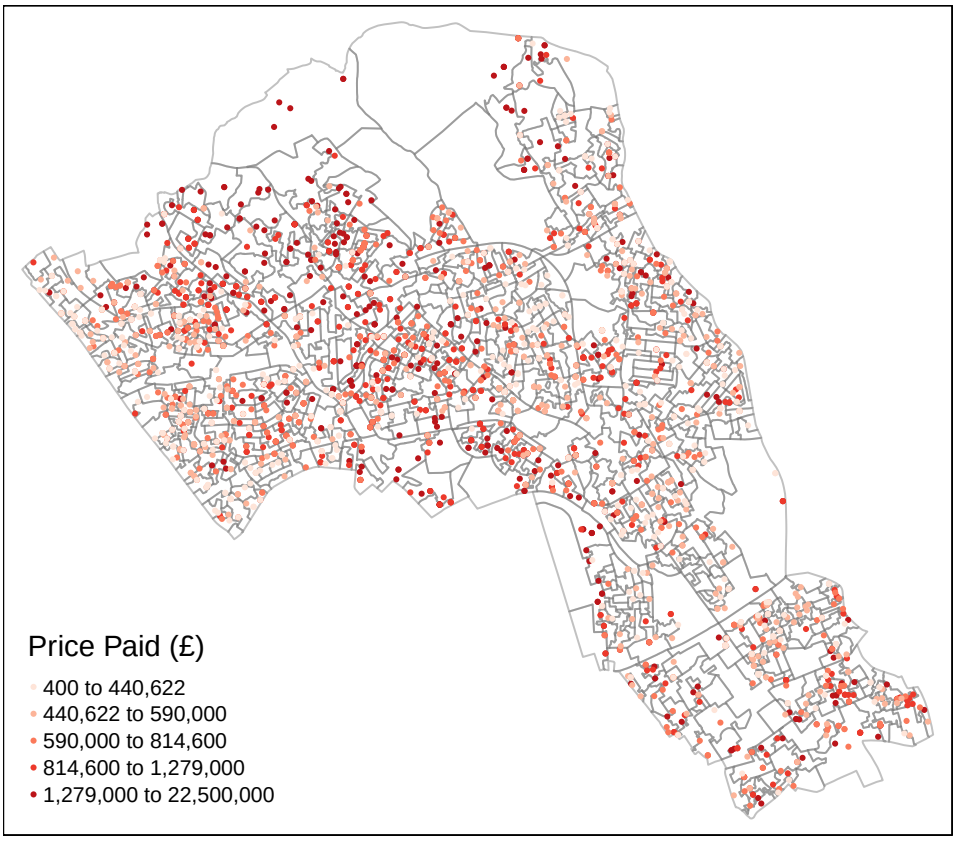
We are now going to use the house price data. We only need to select a few columns to work with. Then we will convert the CSV into a SpatialPointsDataFrame. To do this we will need to set what the data is to be included, what columns contain the x and y coordinates, and what projection system we are using. Notice that it is the same coordinate system as earlier.

```
houses <- houses[,c(1,2,8,9)]

House.Points <-SpatialPointsDataFrame(houses[,3:4], houses,
                                      proj4string = CRS("+init=EPSG:27700"))
```

What we are going to do next is plot a blank base map with polygon boundaries with `tm_shape()`. Then we superimpose our spatial points data frame on top of the blank base map by adding `tm_shape()` as a parameter. We can do this with our data to make a map that helps us visualize housing prices in Camden.

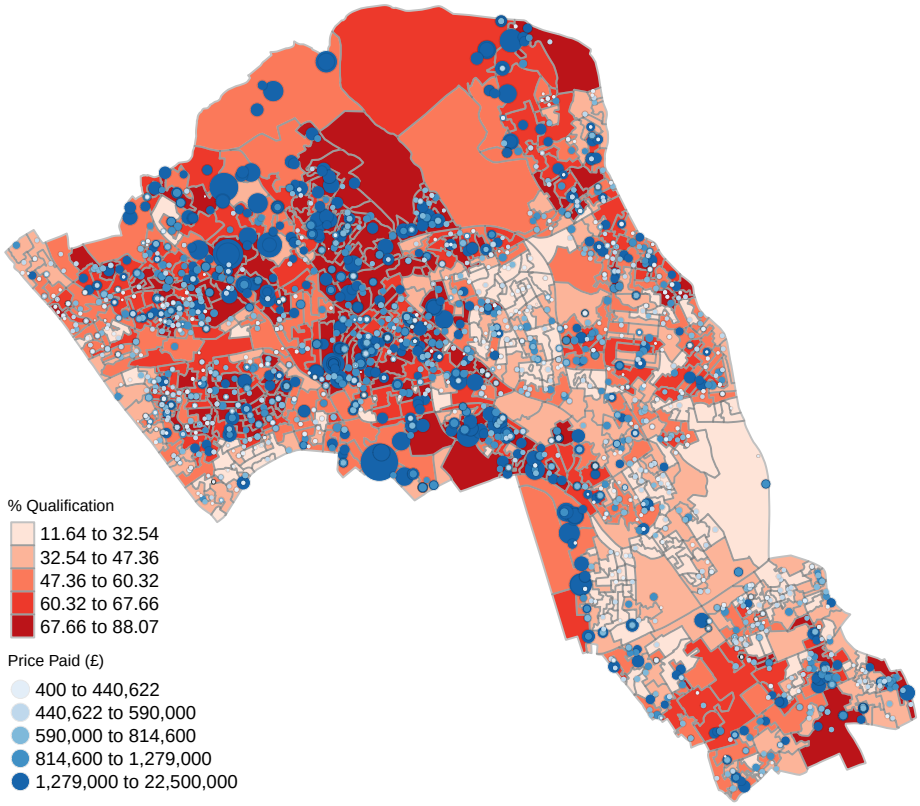
```
tm_shape(OA.Census) +
  tm_borders(alpha=.4) +
  tm_shape(House.Points) +
  tm_dots(col = "Price", scale = 1.5, palette = "Reds", style = "quantile",
          title = "Price Paid (£)")
```



Combining Point Data with Census Data

We can also visualize variables from our census data and point data at the same time. Here we get a look at how much homes cost in areas with greater proportions of qualified individuals.

```
tm_shape(OA.Census) +
  tm_fill("Qualification", palette = "Reds",
          style = "quantile", title = "% Qualification") +
  tm_borders(alpha=.4) +
tm_shape(House.Points) +
  tm_bubbles(size = "Price", col = "Price", palette = "Blues",
             style = "quantile", legend.size.show = FALSE,
             title.col = "Price Paid (£)", border.col = "black",
             border.lwd = 0.1, border.alpha = 0.1) +
tm_layout(legend.text.size = 0.6, legend.title.size = 0.6, frame = FALSE)
```



Exercise 2: Make a map

Make a map that tells us information about unemployment rates and prices paid for homes in different areas of Camden. Choose colors and a layout that make the relationships easy to see, and come up with an appropriate title.

Using R as a GIS: point-in-polygon

We will now further explore some of the spatial analysis functionality of R.

First, make sure we are using the same coordinate reference system for all our files. We want to give the points in our spatial points data frame the attributes of the polygon they fall within. This is called a point-in-polygon operation. Then, bind the census data to our original points to have all our data in a single data frame.

```
proj4string(OA.Census) <- CRS("+init=EPSG:27700")
proj4string(House.Points) <- CRS("+init=EPSG:27700")

pip <- over(House.Points, OA.Census)
df <- cbind(House.Points@data, pip)
```

(The `@data` just lets us work with our class of objects — don't worry too much about it.)

Exercise 3: Make a plot

With our new data frame, `df`, make a plot that compares house prices and local qualification rates. Can you discern any pattern? Maybe a map would be a more useful visualization of the data. Hint: use a `log` transformation of the price variable.

Using R as a GIS: mapping

When it comes to mapping, we may want to work in the opposite direction as our point-in-polygon operation above. That is, we want to measure average house prices in each output area. We can do this using the `aggregate()` function. Then change the column names of the aggregated data, and join the aggregated data back to the `OA.Census` polygon to create a single object.

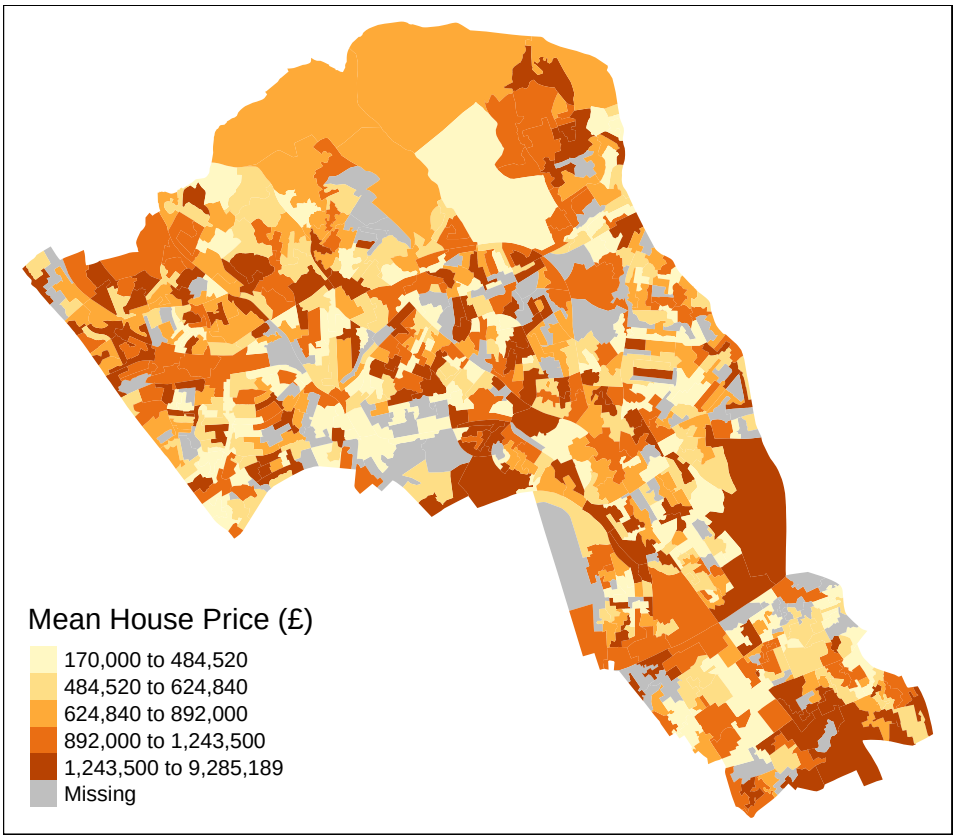
```
OA <- aggregate(df$Price, by = list(df$OA11CD), mean)

names(OA) <- c("OA11CD", "Price")

OA.Census@data <- merge(OA.Census@data, OA, by = "OA11CD", all.x = TRUE)
```

We can now create a map that tells us the average price at which houses were sold in each area in 2015. If no houses were sold in a given area, the data will appear as "missing."

```
tm_shape(OA.Census) + tm_fill(col = "Price", style = "quantile",
                              title = "Mean House Price (£)")
```



Using R as a GIS: linear regression

Recall how to run a simple linear regression with the `lm()` function. We can now run a linear model to see the relationship between our unemployment variable and our new average house price variable.

```
lm <- lm(OA.Census@data$Price ~ OA.Census@data$Unemployed)

summary(lm)
```

```
##
## Call:
## lm(formula = OA.Census@data$Price ~ OA.Census@data$Unemployed)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -977841 -364618 -138499  145581 8053263
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  1434296      58128  24.675   <2e-16 ***
## OA.Census@data$Unemployed  -110798      11754  -9.426   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 748300 on 652 degrees of freedom
##   (95 observations deleted due to missingness)
## Multiple R-squared:  0.1199, Adjusted R-squared:  0.1186
## F-statistic: 88.85 on 1 and 652 DF,  p-value: < 2.2e-16
```

Exercise 4: Make a map, run a regression

Make a map that displays information on housing prices, unemployment rates, and rates of qualification. Use any of the techniques we have covered. Then use a multiple regression to see how the latter two variables correlate with housing prices.

Kernel density estimation

Using a kernel density estimation allows us to get a more "continuous" idea of the density of observations in space. The following code gives us a kernel density estimation for our housing price data. The general process can be broken down:

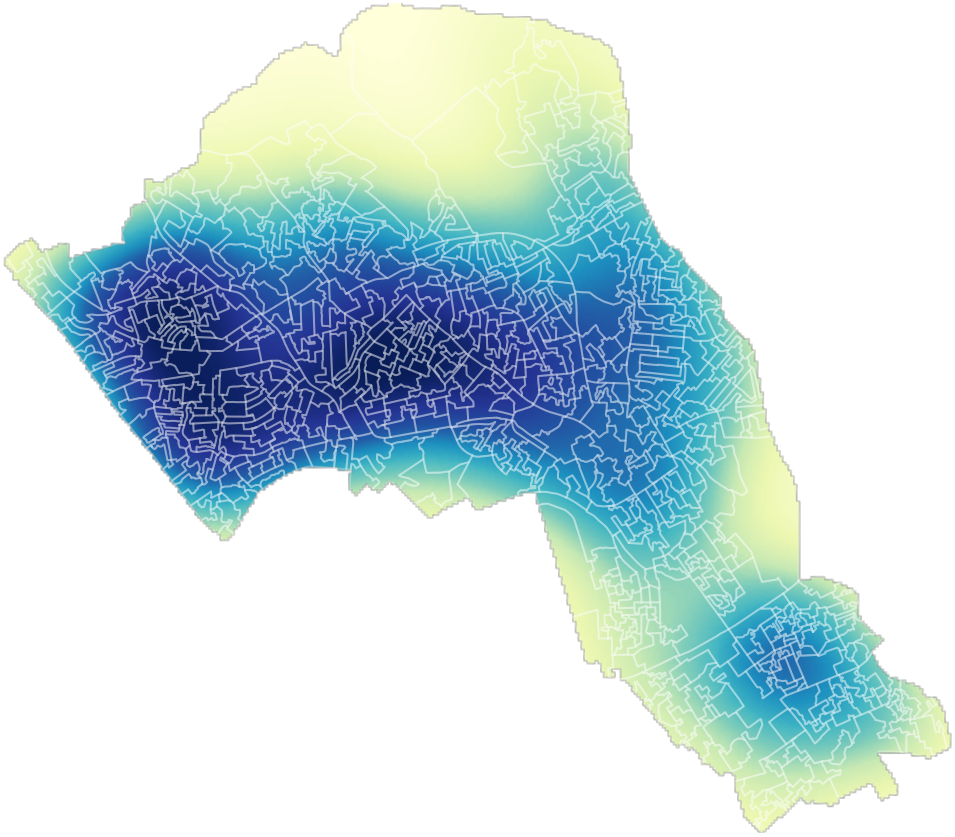
1. Run the kernel desity estimation
2. Convert the data to raster (think of it as pixels v. the polygon "vectors" we have been using so far), and set the appropriate projection
3. Create a bounding box based on the boundary of the polygon we want to overlay the data on eventually (for us, this is the `Output.Areas` polygon)
4. Mask the raster by the output area polygon (this is like Photoshopping the raster to the precise dimensions of the polygon)
5. Map the raster with the polygon boundaries

See on the next page what this looks like.

```
kde.output <- kernelUD(House.Points, h="href", grid = 1000)
kde <- raster(kde.output)
projection(kde) <- CRS("+init=EPSG:27700")

bounding_box <- bb(Output.Areas)
masked_kde <- mask(kde, Output.Areas)

tm_shape(masked_kde, bbox = bounding_box) + tm_raster("ud", style = "quantile",
                                                      n = 100,
                                                      legend.show = FALSE,
                                                      palette = "YlGnBu") +
tm_shape(Output.Areas) + tm_borders(alpha=.3, col = "white") +
tm_layout(frame = FALSE)
```



Exercise 5: Add to the kernel density estimation

Recreate the following map. It gives us an idea of how many high-priced houses are in low-density areas, and vice versa.