Practical 7: Using R as a GIS

An Introduction to Spatial Data Analysis and Visualisation in R - Guy Lansley & James Cheshire (2016)

This tutorial is intended to provide a demonstration of some of the basic spatial functionality of R by taking you through a small number of commonly employed techniques. Data for the practical can be downloaded from the **Introduction to Spatial Data Analysis and Visualisation in R** (https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r) homepage.

In this tutorial we will:

- Run a point-in-polygon operation
- Create buffers
- Add backing maps from Google
- reproject spatial data
- Create interactive maps with tmap

First, we must set the working directory and load the practical data.

```
# Set the working directory
setwd("C:/Users/Guy/Documents/Teaching/CDRC/Practicals")

# Load the data. You may need to alter the file directory
Census.Data <-read.csv("practical_data.csv")
```

We will also need to load the spatial data files from the previous tutorials.

```
# load the spatial libraries
library("sp")
library("rgdal")
library("rgeos")

# Load the output area shapefiles
Output.Areas <- readOGR(".", "Camden_oa11")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "Camden_oa11"
## with 749 features
## It has 1 fields
```

```
# join our census data to the shapefile
OA.Census <- merge(Output.Areas, Census.Data, by.x="OA11CD", by.y="OA")

# load the houses point files
House.Points <- readOGR(".", "Camden_house_sales")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "Camden_house_sales"
## with 2547 features
## It has 4 fields
```

We can now commence our operations.

Point-in-polygon

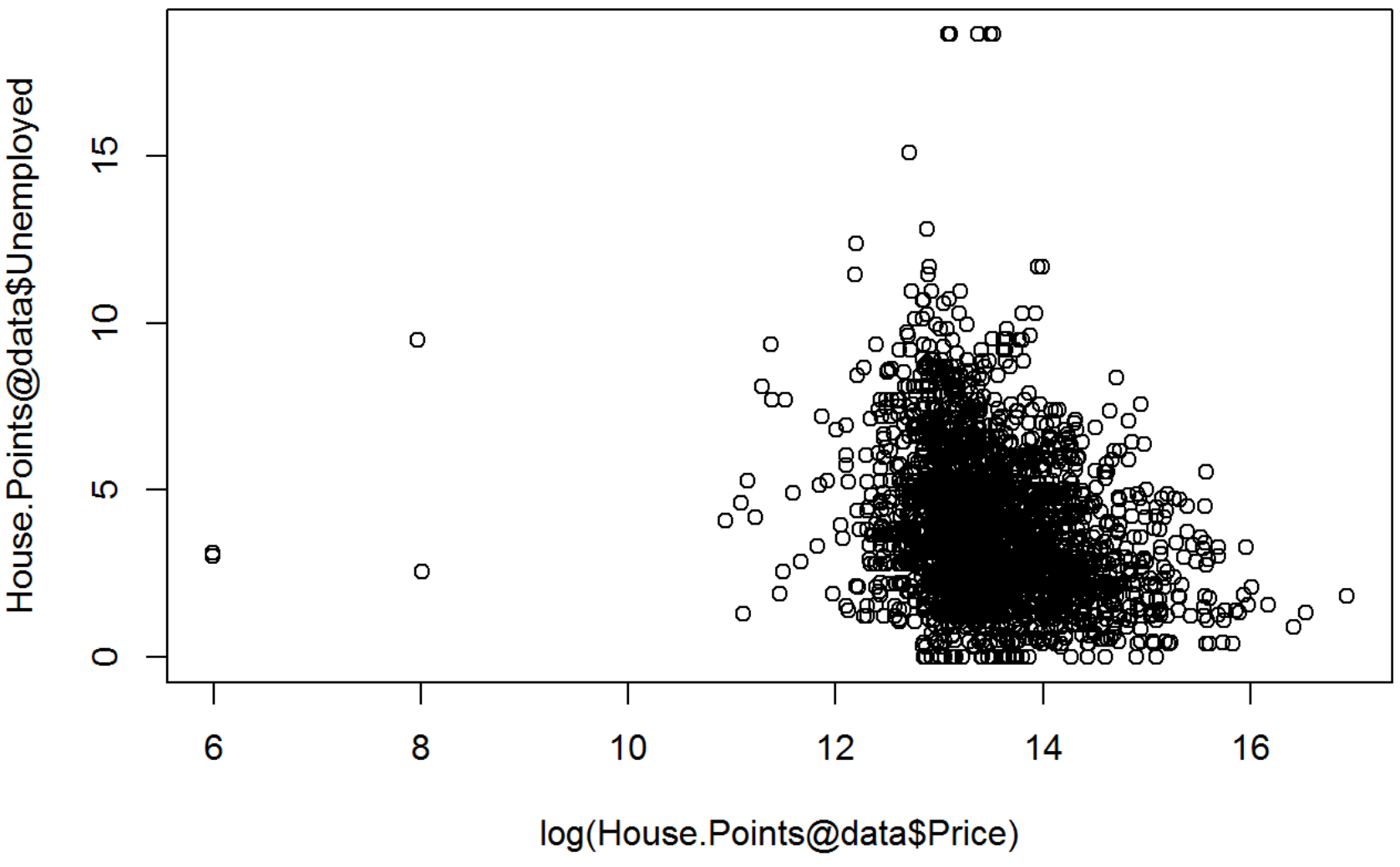
Firstly, we will aim to aggregate our point data into the Output Area polygons using a point in polygon (http://www.spatialanalysisonline.com/HTML/?point_object_in_polygon_pip.htm) operation.

As we are commencing a spatial operation, both files need to be projected using the same coordinate reference system. In this case, we have ensured that both spatial files have been set to British National Grid (27700).

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

With the projections set, it is now possible to assign each house point the characteristics of the output area polygon it falls within.

```
# point in polygon. Gives the points the attributes of the polygons that they are
in
pip <- over(House.Points, OA.Census)

# need to bind the census data to our original points
House.Points@data <- cbind(House.Points@data, pip)

View(House.Points@data)

# it is now possible to plot the house prices and local unemployment rates
plot(log(House.Points@data$Price), House.Points@data$Unemployed)
```

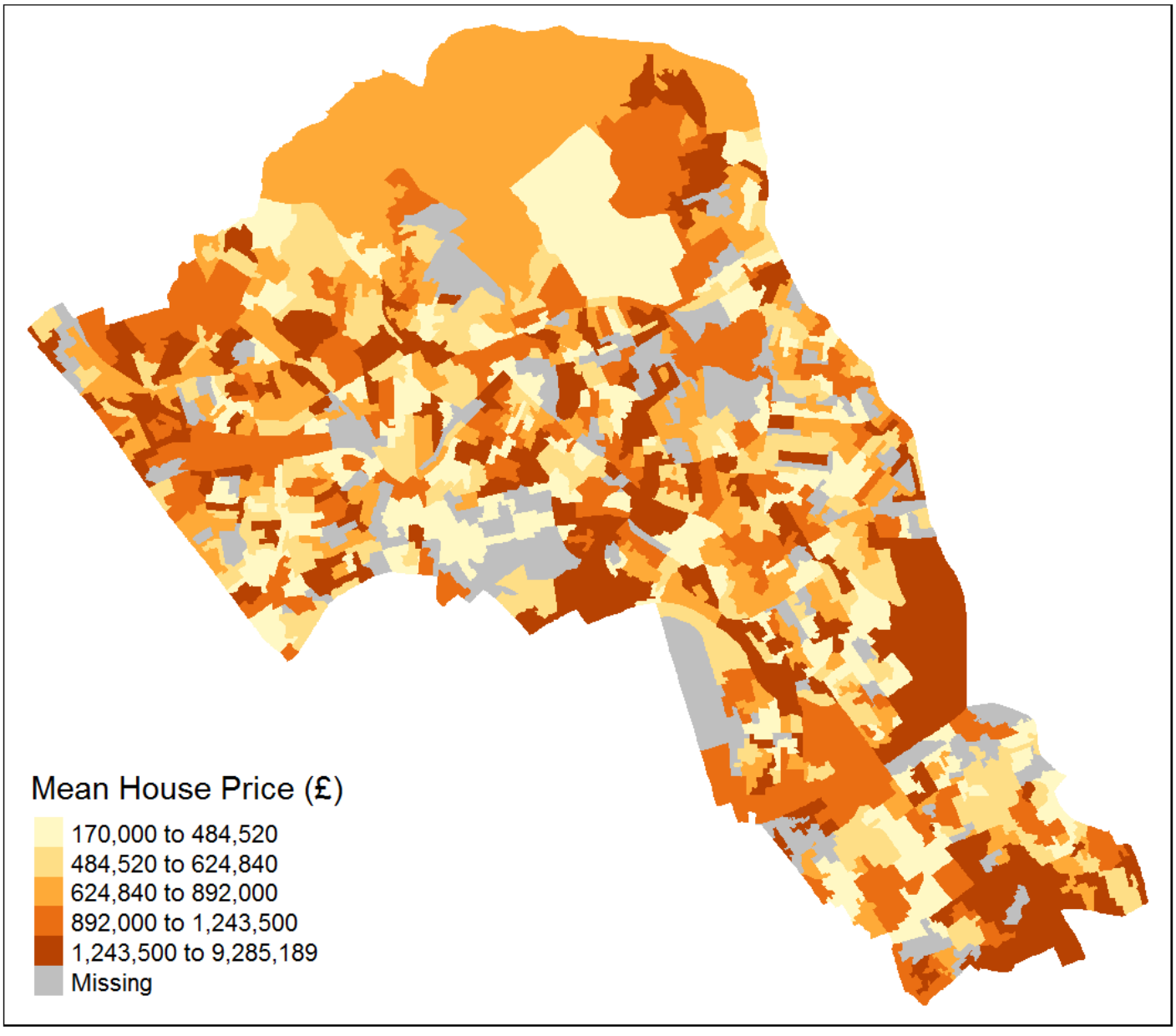


It is also useful to measure the average house prices for each output area. We don't need to run another point in polygon operation, instead, we can reaggregate the data by the **OA11CD** column so every output area has one record. Using the `aggregate()` function we can decide what numbers are returned from our house prices (i.e. mean, sum, median).

```
# first we aggregate the house prices by the OA11CD (OA names) column, we ask for
the mean for each OA
OA <- aggregate(House.Points@data$Price, by = list(House.Points@data$OA11CD), mean
)

# change the column names of the aggregated data
names(OA) <- c("OA11CD", "Price")

# join the aggregated data back to the OA.Census polygon
OA.Census@data <- merge(OA.Census@data, OA, by = "OA11CD", all.x = TRUE)
```

We can now map the data using tmap. We will have missing data where there are output areas where no houses were sold in 2015.

```
library(tmap)

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



It is also possible to now run a linear model between our unemployment variable from the 2011 Census and our new average house price variable.

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

```
##
## 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      58129  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
```

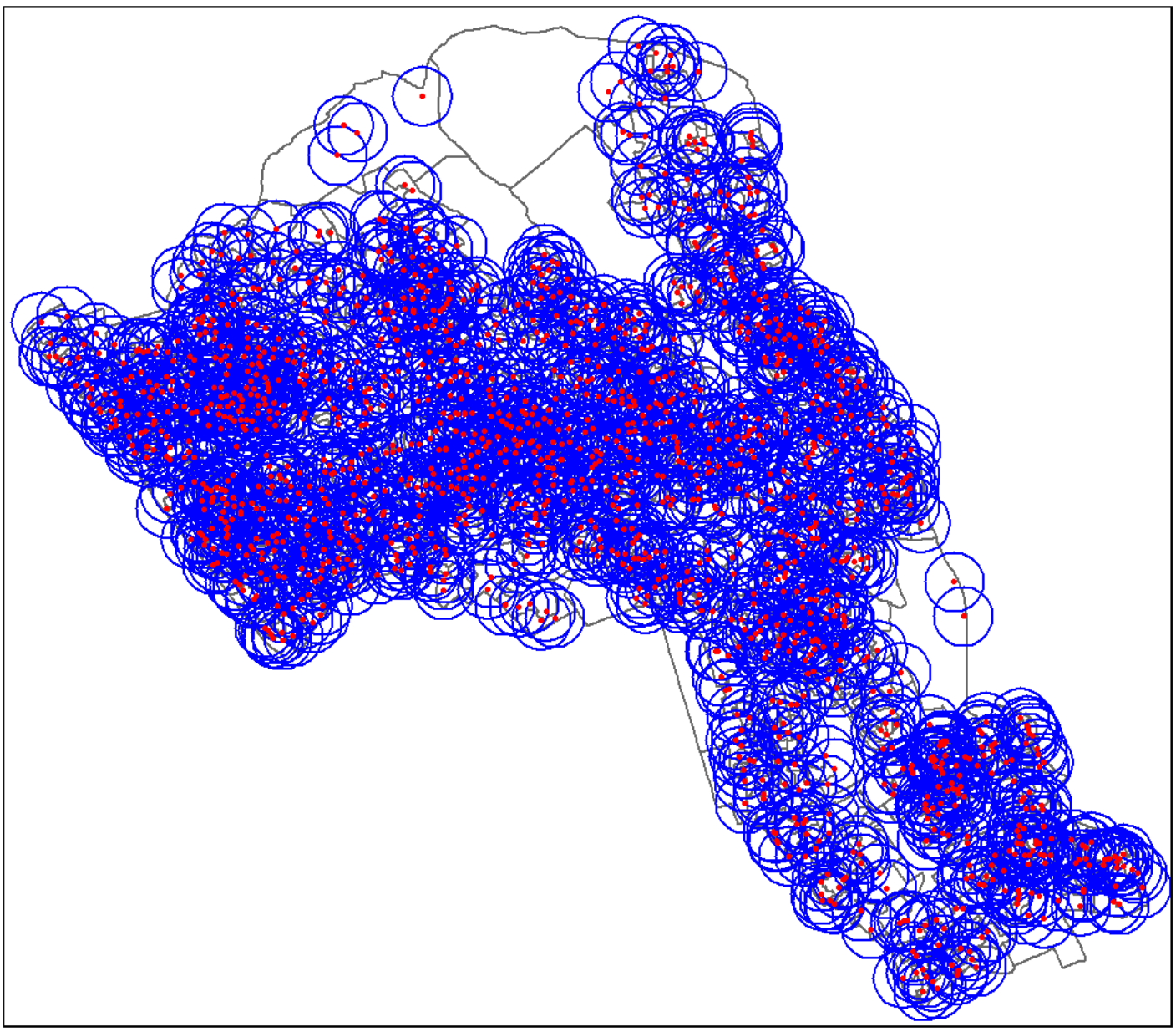
Buffers

A second technique we will demonstrate is buffering (http://www.spatialanalysisonline.com/HTML/?buffering.htm). This is a GIS process by which we create linear catchments for each data point based on distance. This simple technique is commonly used to determine which areas are proximal to certain objects. Here we use the `gBuffer()` function from the rgeos package.

```
# create 200m buffers for each house point
house_buffers <- gBuffer(House.Points, width = 200, byid = TRUE)
```

We can plot these in tmap.

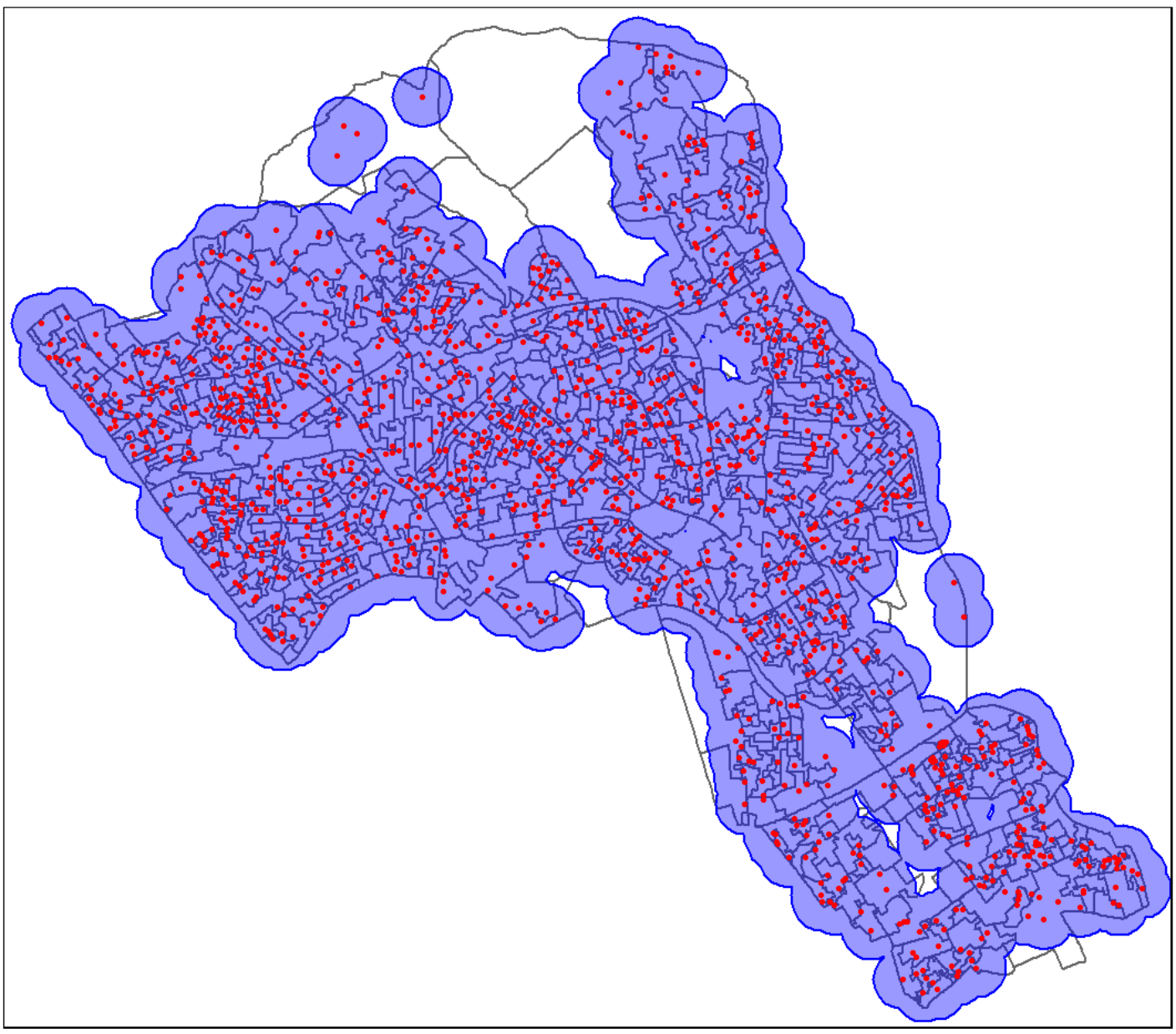
```
# map in tmap
tm_shape(OA.Census) + tm_borders() +
tm_shape(house_buffers) + tm_borders(col = "blue") +
tm_shape(House.Points) + tm_dots(col = "red")
```



Union

We can merge all of the buffers together using a process known as a union. This process will join all intersecting geometries.

```
# merges the buffers
union.buffers <- gUnaryUnion(house_buffers)

# map in tmap
tm_shape(OA.Census) + tm_borders() +
tm_shape(union.buffers) + tm_fill(col = "blue", alpha = .4) + tm_borders(col = "bl
ue") +
tm_shape(House.Points) + tm_dots(col = "red")
```



Adding Backing Maps

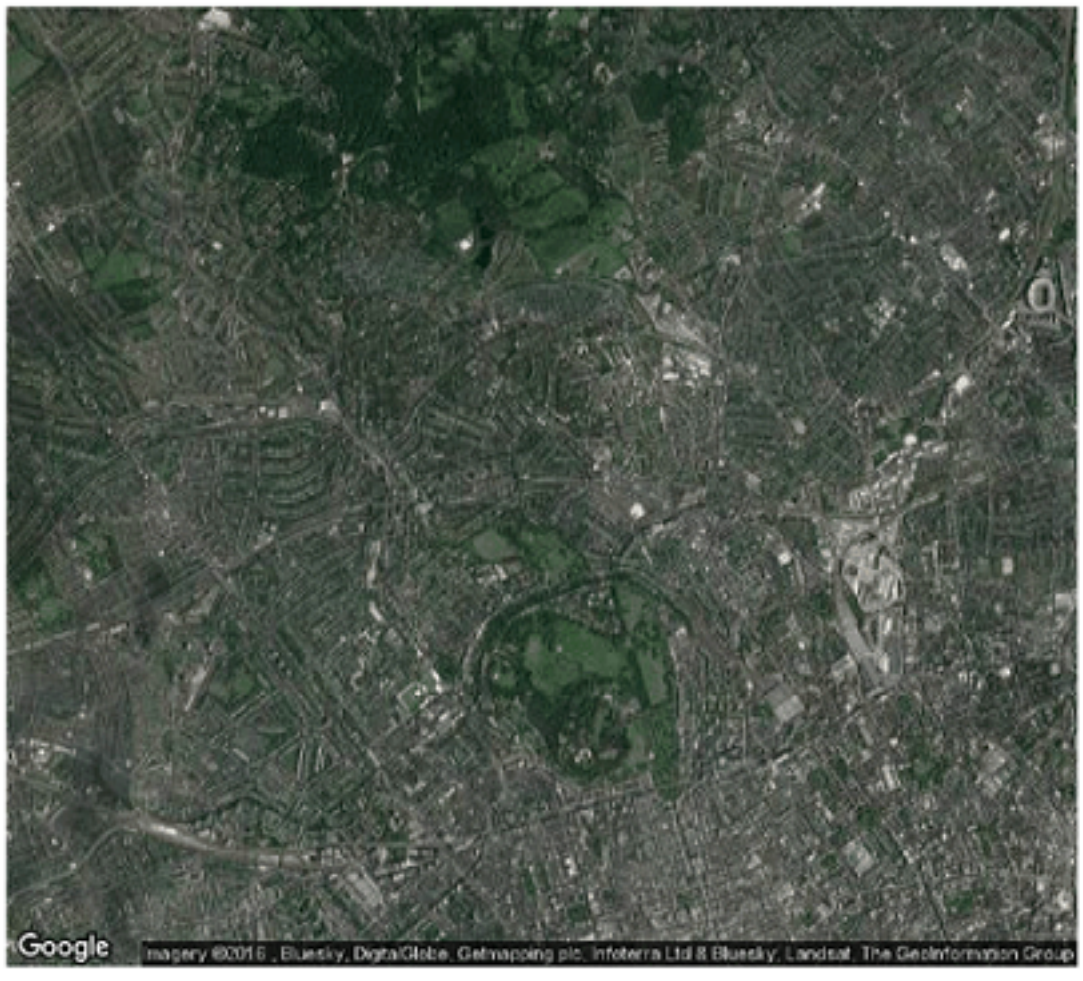
Adding backing maps is sometimes more complicated than it first seems as each mapping provider may use a different coordinate reference system. In this example, we will download backing maps from Google and project our data on to them.

```
library(raster)
library(dismo)

google.map <- gmap("Camden, London", type = "satellite")
```

```
## OVER_QUERY_LIMIT:Camden,London
```

```
## [1] "try 2 ..."
## [1] "try 3 ..."
```

You can plot these with `plot(google.map)`



We can also change the type of map we wish to download, and then write it to our file space. For instance.

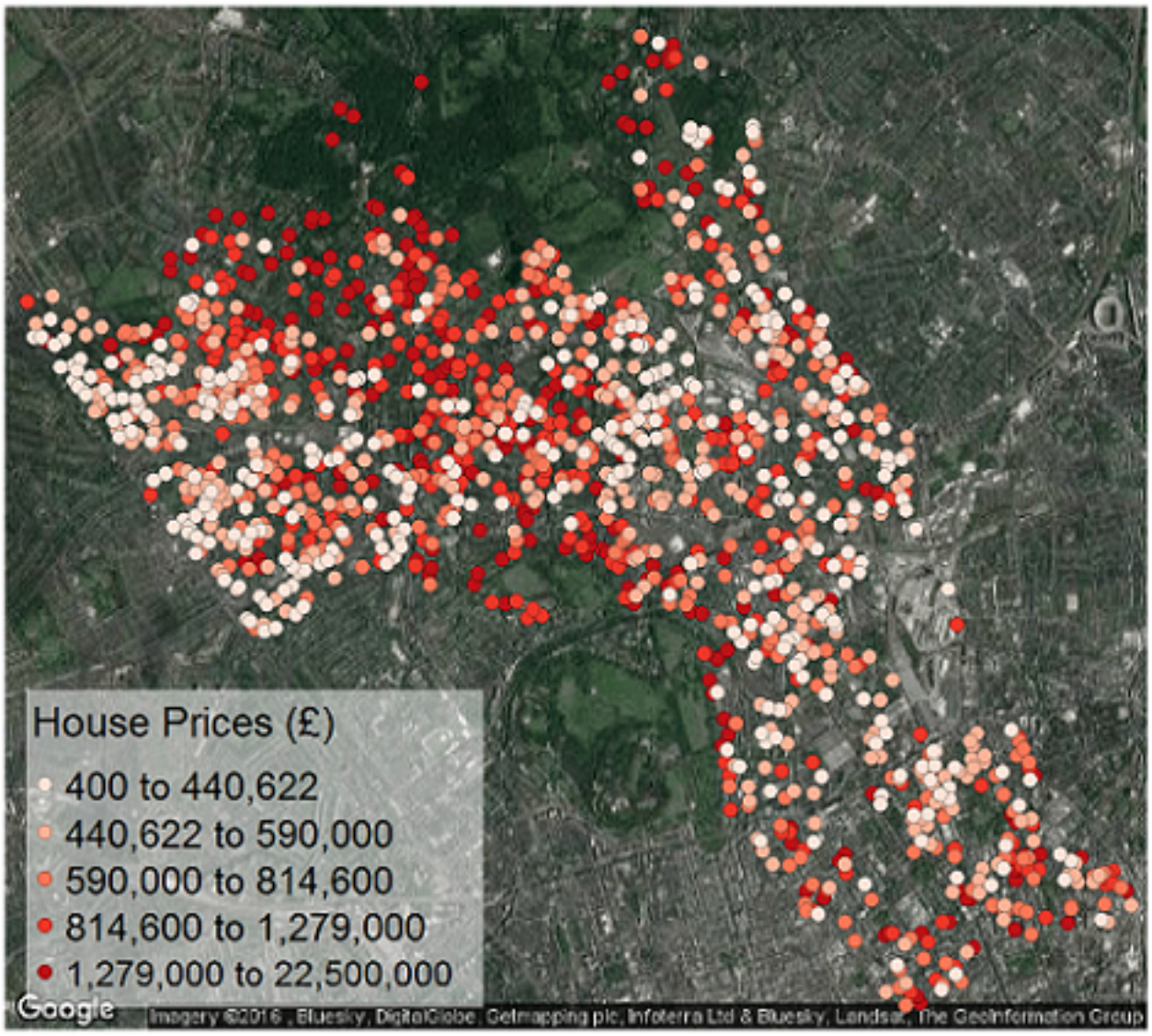
```
#save the map
google.map <- gmap("Camden, London", type = "roadmap", filename = "Camden.gmap")
```

Next, we want to plot our house points onto this map. However, our house data is projected via the British National Grid coordinate system, while Google uses the web Mercator projection. We can transform one of the projections using the code below.

```
# reproject the House.Points to the WGS84 projection
CRS.new <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84")
reprojected.houses <- spTransform(House.Points, CRS.new)
```

We can now map both the base map and reprojected house points in tmap. We need to treat the backing map as a raster image.

```
# maps the base and reprojected house points in tmap
tm_shape(google.map) + tm_raster() +
tm_shape(reprojected.houses) + tm_dots(col = "Price", style = "quantile", scale =
2.5, palette = "Reds", title = "House Prices (£)", border.col = "black", border.lw
d = 0.1, border.alpha = 0.4)  +
tm_layout(legend.position = c("left", "bottom"),  legend.text.size = 1.1, legend.t
itle.size = 1.4, frame = FALSE, legend.bg.color = "white", legend.bg.alpha = 0.5)
```



Creating Interactive Maps

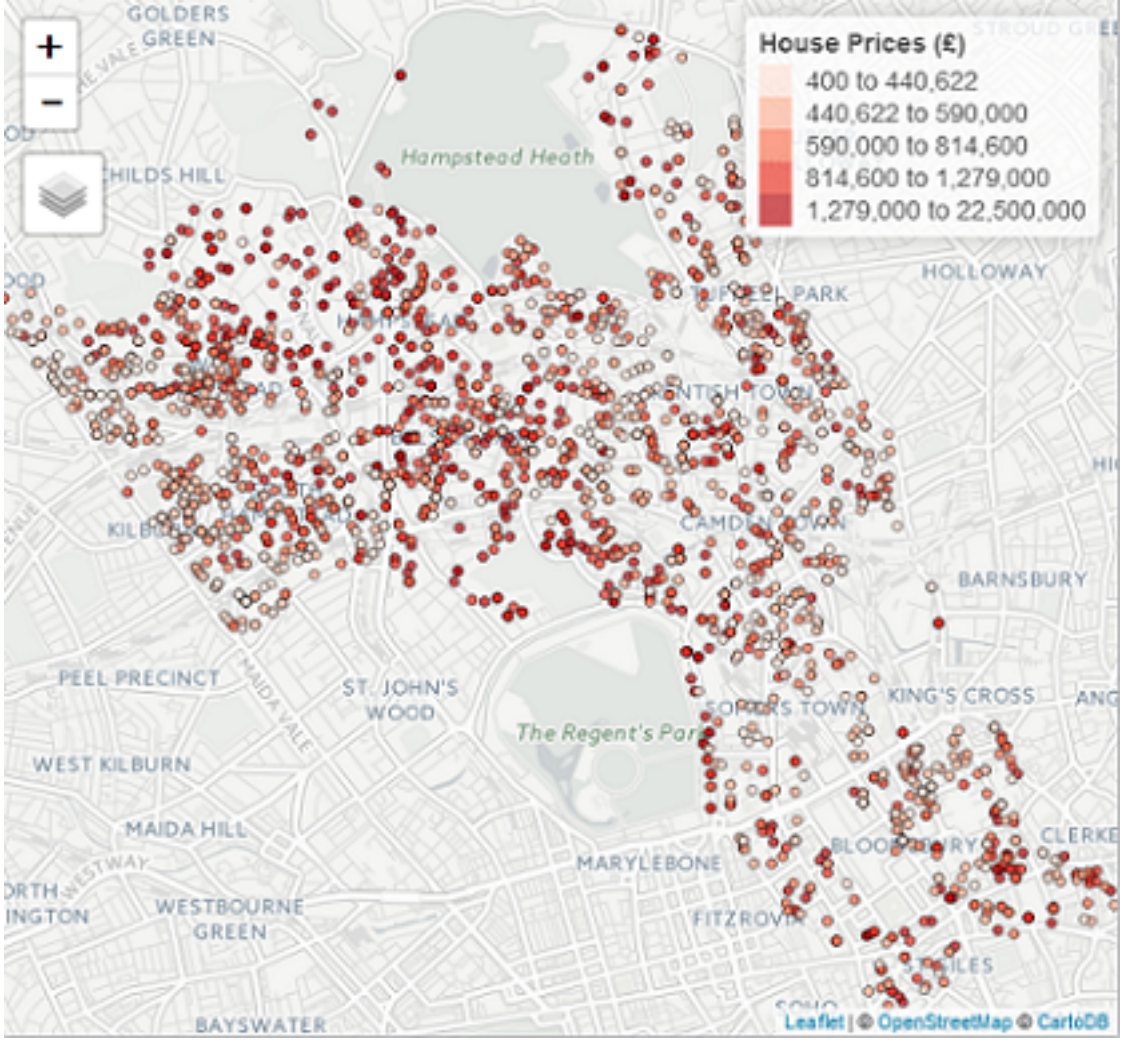
We have now sussed the basics of mapping in R using the tmap package. However, all of the maps produced so far have been static images. It is possible to create interactive 'slippy' maps through which users can zoom in and out, and even turn layers on and off. This can be very easily done in tmap using the same techniques as before. Please note that this technique only works on the most recent editions of RStudio.

The first thing we need to do is to tell R to switch our tmap mode from `plot` to `view`.

```
# interactive maps in tmap
library(leaflet)

# turns view map on
tmap_mode("view")
```

With view mode turned on, every map we now produce with tmap will be produced as an interactive slippy map. First, we will map our house price data (note we do not need to include any attributes on layout anymore).

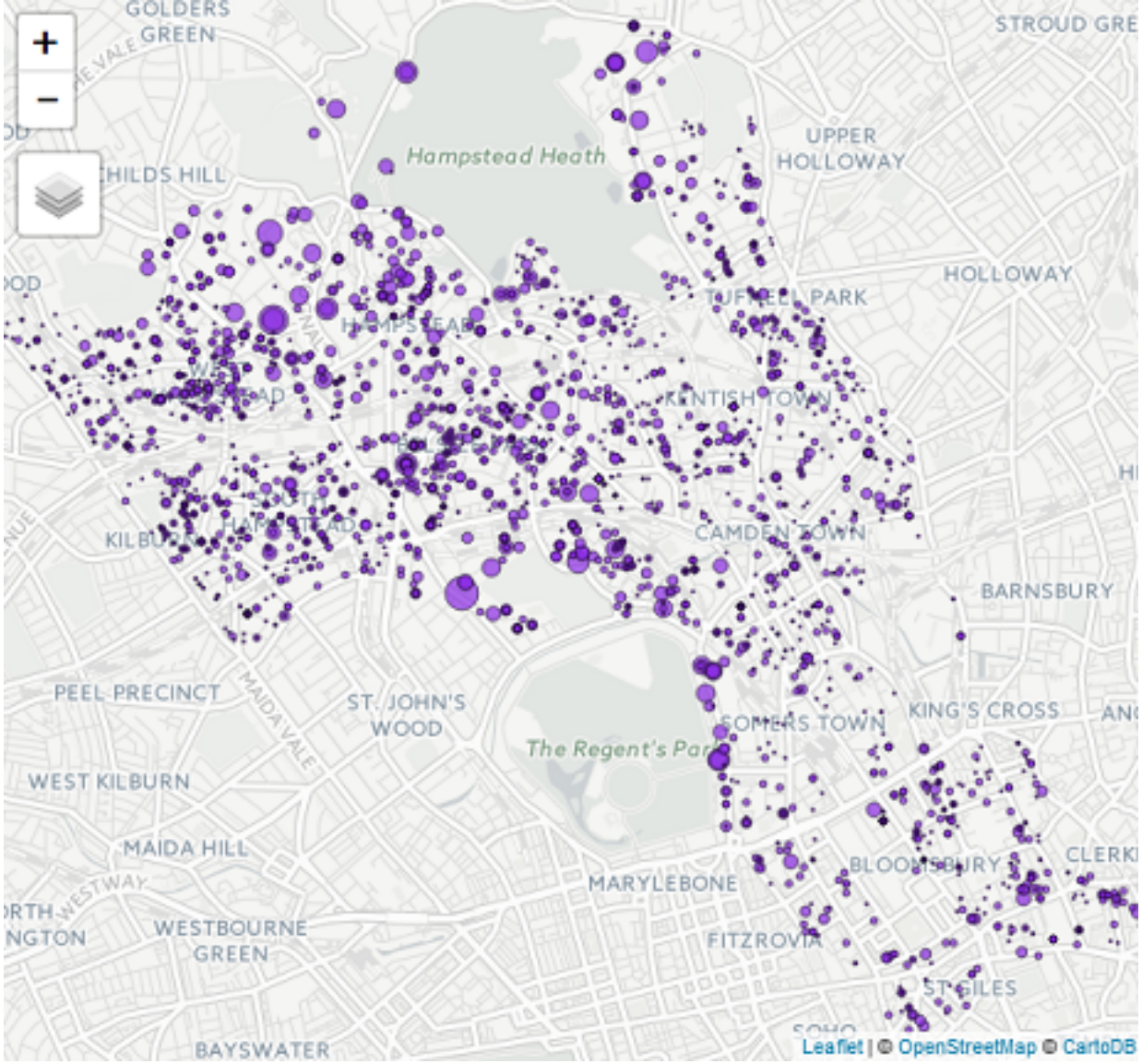
```
tm_shape(House.Points) + tm_dots(title = "House Prices (£)", border.col = "black",
border.lwd = 0.1, border.alpha = 0.2, col = "Price", style = "quantile", palette =
"Reds")
```



The map is interactive. You can zoom in using the controls in the top left corner of the viewer window, you can also change the backing map and turn off layers we have included. In addition, you can click on data in the map to view the attributes for that point. The default base map is a grey map system known as *CartpDB.Positron*. The advantage of the grey shades is that they won't clash with our coloured data. You can also click on the layer option to switch to the other map options which include *OpenStreetMap*.

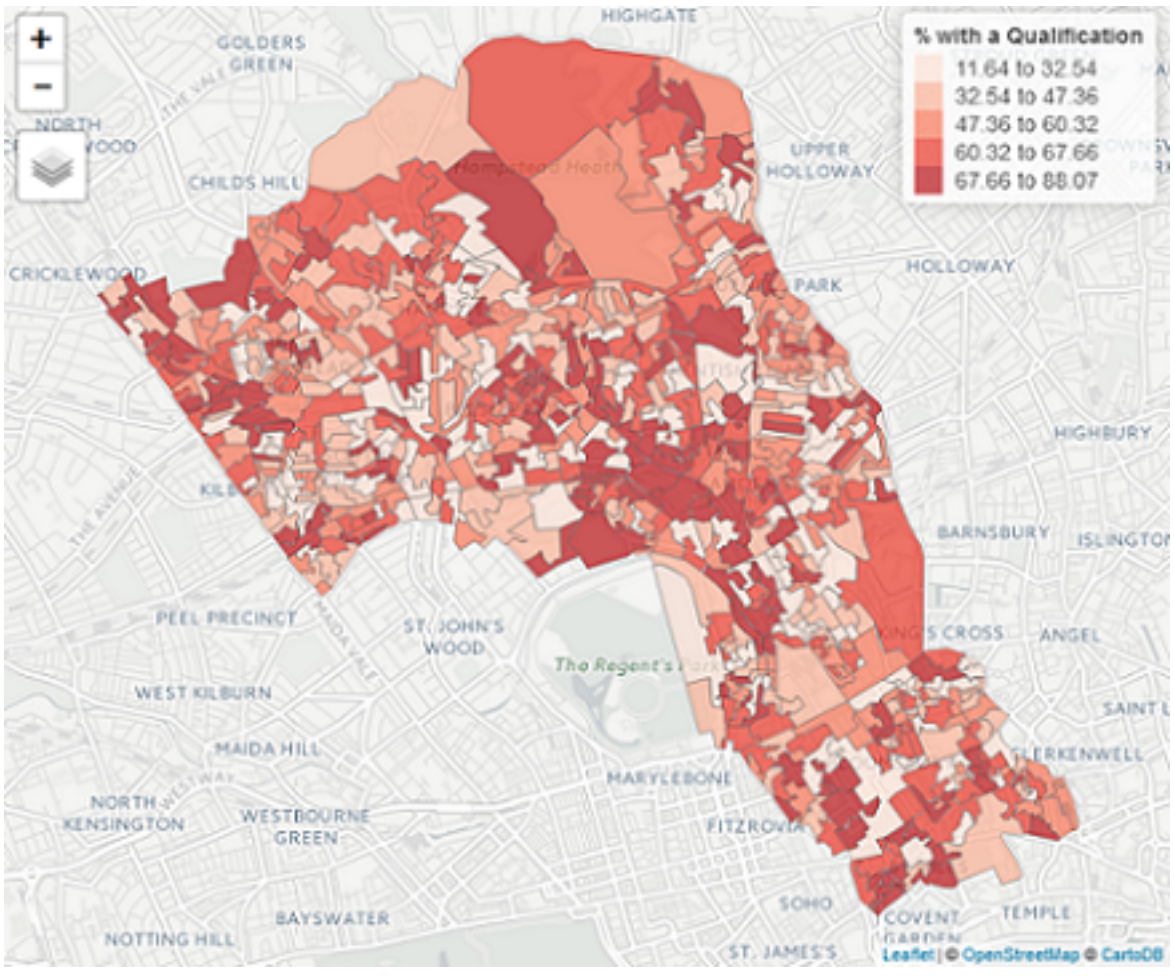
The interactive version of tmap works with all of the tmap functions. For example, we can make bubble plots using `tm_bubbles()` as we did in the previous practical.

```
tm_shape(House.Points) + tm_bubbles(size = "Price", title.size = "House Prices (£)
", border.col = "black", border.lwd = 0.1, border.alpha = 0.4, legend.size.show =
TRUE)
```

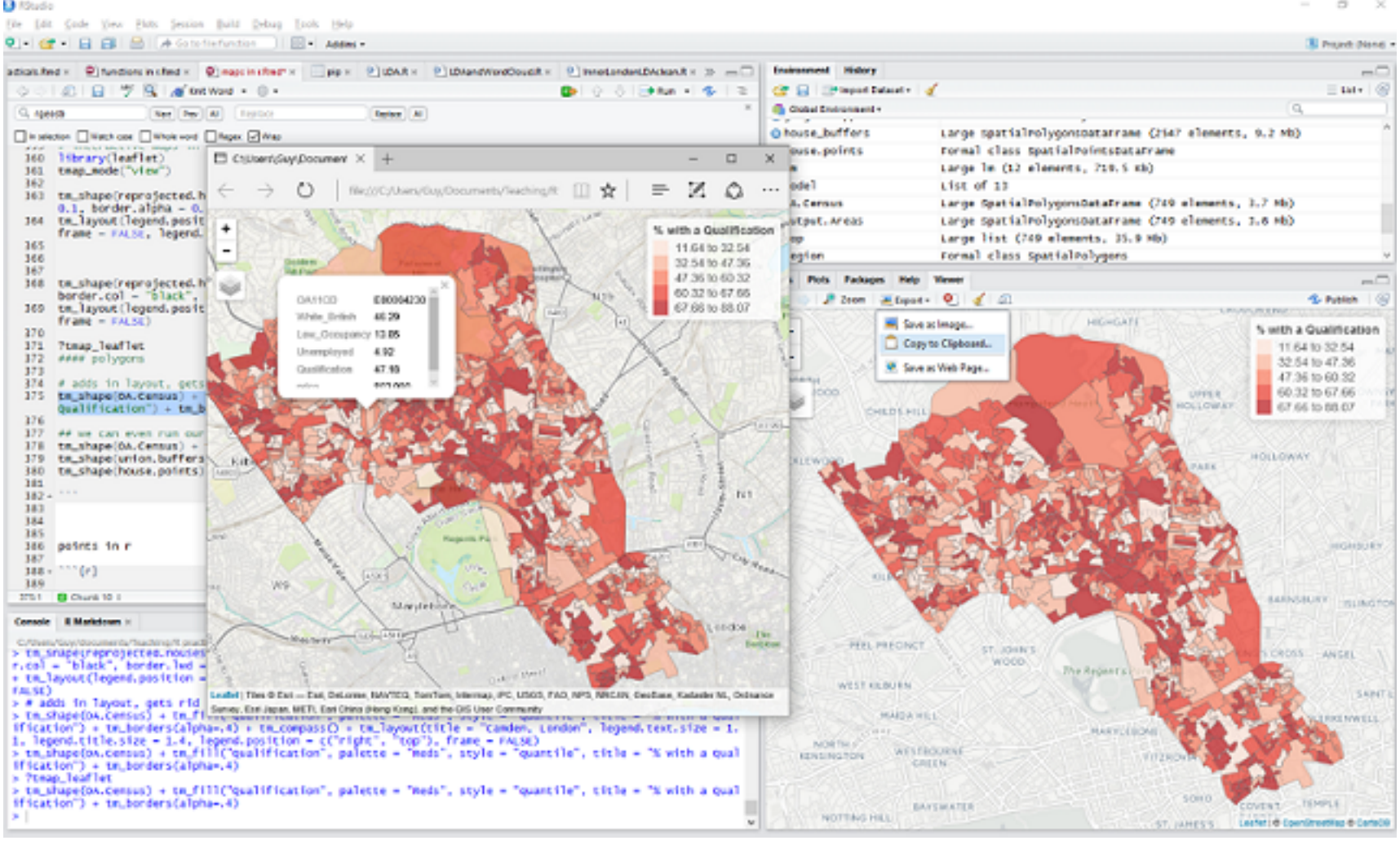


We can also map polygons…

```
tm_shape(OA.Census) + tm_fill("Qualification", palette = "Reds", style = "quantile
", title = "% with a Qualification") + tm_borders(alpha=.4)
```



Finally, if you wish to export your map as an interactive webpage click on export in the plots window go click on **Save as Web Page…**. The html file can either be viewed locally on a web browser, or the code can be inputted into a website so it can be displayed on the internet as an interactive widget.



To turn tmap back to the plot view simply run `tmap_mode("plot")` again.

If you are interested in more advanced methods for customising interactive maps, please explore the leaflet package in R.

The rest of the online tutorials in this series can be found at: https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r (https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r)