Points

Dani Arribas-Bel

2016-03-17

This session[1] is based on the following references, which are great follow-up's on the topic:

- Lovelace and Cheshire (2014) is a great introduction.
- Chapter 6 of Brunsdon and Comber (2015), in particular subsections 6.3 and 6.7.
- Bivand, Pebesma, and Gómez-Rubio (2013) provides an in-depth treatment of spatial data in R.

This tutorial is hosted as a GitHub repository and you can access it in a few ways:

- As a download of a `.zip` file that contains all the materials.
- As an html website.
- As a pdf document
- As a GitHub repository.

Dependencies

This tutorial relies on the following libraries that you will need to have installed on your machine to be able to interactively follow along[2]. Once installed, load them up with the following commands:

[2] You can install package `mypackage` by running the command `install.packages("mypackage")` on the R prompt or through the `Tools --> Install Packages...` menu in RStudio.

```
# Layout
library(tufte)
# Spatial Data management
library(rgdal)
# Pretty graphics
library(ggplot2)
# Pretty maps
library(ggmap)
# Various GIS utilities
library(GISTools)
# For all your interpolation needs
library(gstat)
# For data manipulation
library(plyr)
```

Before we start any analysis, let us set the path to the directory where we are working. We can easily do that with `setwd()`. Please replace in the following line the path to the folder where you have placed this file -and where the `house_transactions` folder with the data lives.

```
#setwd('~/AAA/Documents/teaching/u-lvl/2016/envs453/code')
#setwd('/media/dani/baul/AAA/Documents/teaching/u-lvl/2016/envs453/code')
setwd('.')
```

Data

For this session, we will use a subset of residential property transaction data for the city of Liverpool. These are provided by the Land Registry (as part of their Price Paid Data) but have been cleaned and re-packaged by Dani Arribas-Bel.

Let us start by reading the data, which comes in a shapefile:

```
db <- readOGR(dsn = 'house_transactions', layer = 'liv_house_trans')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "house_transactions", layer: "liv_house_trans"
## with 6324 features
## It has 18 fields
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
```

We can then examine the elements of the object with the `summary` method:

```
summary(db)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##               min    max
## coords.x1 333536 345449
## coords.x2 382684 397833
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=49 +lon_0=-2
## +k=0.9996012717 +x_0=400000
## +y_0=-100000 +datum=OSGB36 +units=m
## +no_defs +ellps=airy
## +towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894]
## Number of points: 6324
## Data attributes:
##       pcds
##  L1 6LS : 126
```

```
## L8 5TE :  63
## L1 5AQ :  34
## L24 1WA:  31
## L17 6BT:  26
## L3 1EE :  24
## (Other):6020
##                                       id
## {00029226-80EF-4280-9809-109B8509656A}:   1
## {00041BD2-4A07-4D41-A5AE-6459CD5FD37C}:   1
## {0005AE67-9150-41D4-8D56-6BFC868EECA3}:   1
## {00183CD7-EE48-434B-8A1A-C94B30A93691}:   1
## {003EA3A5-F804-458D-A66F-447E27569456}:   1
## {00411304-DD5B-4F11-9748-93789D6A000E}:   1
## (Other)                               :6318
##     price                     trans_date
## Min.   :    1000   2014-06-27 00:00: 109
## 1st Qu.:   70000   2014-12-19 00:00: 109
## Median :  110000   2014-02-28 00:00: 105
## Mean   :  144310   2014-10-31 00:00:  95
## 3rd Qu.:  160000   2014-03-28 00:00:  94
## Max.   :26615720   2014-11-28 00:00:  94
##                    (Other)         :5718
## type     new      duration      paon
## D: 505   N:5495   F:3927   3      : 203
## F:1371   Y: 829   L:2397   11     : 151
## O: 119                     14     : 148
## S:1478                     5      : 146
## T:2851                     4      : 140
##                            8      : 128
##                            (Other):5408
##          saon                   street
## FLAT 2     :  25   CROSSHALL STREET: 133
## FLAT 3     :  25   STANHOPE STREET :  63
## FLAT 1     :  24   PALL MALL       :  47
## APARTMENT 4:  23   DUKE STREET     :  41
## APARTMENT 2:  21   MANN ISLAND     :  41
## (Other)    : 893   OLD HALL STREET :  39
## NA's       :5313   (Other)         :5960
##         locality           town
## WAVERTREE   : 126   LIVERPOOL:6324
## MOSSLEY HILL: 102
## WALTON      :  88
## WEST DERBY  :  71
## WOOLTON     :  66
```

```
##  (Other)     : 548
##  NA's        :5323
##       district           county     ppd_cat
##  KNOWSLEY :  12   MERSEYSIDE:6324   A:5393
##  LIVERPOOL:6311                     B: 931
##  WIRRAL   :   1
##
##
##
##
##  status         lsoa11          LSOA11CD
##  A:6324   E01033762: 144   E01033762: 144
##           E01033756:  98   E01033756:  98
##           E01033752:  93   E01033752:  93
##           E01033750:  71   E01033750:  71
##           E01006518:  68   E01006518:  68
##           E01033755:  65   E01033755:  65
##           (Other)  :5785   (Other)  :5785
```

See how it contains several pieces, some relating to the spatial information, some relating to the tabular data attached to it. We can access each of the separately if we need it. For example, to pull out the names of the columns in the `data.frame`, we can use the `@data` appendix:

```
colnames(db@data)
```

```
##  [1] "pcds"       "id"         "price"
##  [4] "trans_date" "type"       "new"
##  [7] "duration"   "paon"       "saon"
## [10] "street"     "locality"   "town"
## [13] "district"   "county"     "ppd_cat"
## [16] "status"     "lsoa11"     "LSOA11CD"
```

The rest of this session will focus on two main elements of the shapefile: the spatial dimension (as stored in the point coordinates), and the house price values contained in the `price` column. To get a sense of what they look like first, let us plot both. We can get a quick look at the non-spatial distribution of house values with the following commands:

```
# Create the histogram
hist <- qplot(data=db@data,x=price,
              bins=150)
# Change the background to go with the HTML color
histplot <- hist +
```

```
  theme(plot.background = element_rect(fill = '#fffff8'))
histplot
```

This basically shows there is a lot of values concentrated around the lower end of the distribution but a few very large ones. A usual transformation to *shrink* these differences is to take logarithms:

```
# Create log and add it to the table
logpr <- log(db@data$price)
db@data['logpr'] <- logpr
# Create the histogram
hist <- qplot(x=logpr,
              bins=50)
# Change the background to go with the HTML color
histplot <- hist +
  theme(plot.background = element_rect(fill = '#fffff8'))
histplot
```

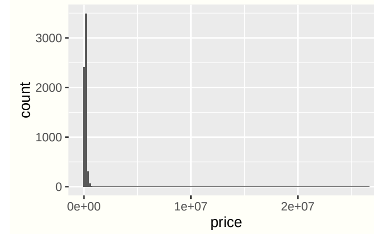


Figure 1: Raw house prices in Liverpool

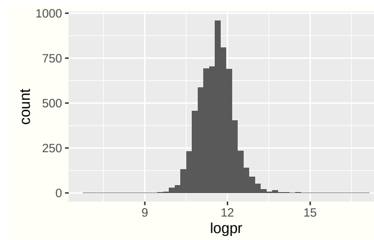


Figure 2: Log of house price in Liverpool

To obtain the spatial distribution of these houses, we need to turn away from the `@data` component of `db`. The easiest, quickest (and also dirtiest) way to get a sense of what the data look like over space is using `plot`:

```
plot(db)
```

Figure 3: Spatial distribution of house transactions in Liverpool

KDE

Kernel Density Estimation (KDE) is a technique that creates a *continuous* representation of the distribution of a given variable, such as house prices. Although theoretically it can be applied to any dimension, usually, KDE is applied to either one or two dimensions.

One-dimensional KDE

KDE over a single dimension is essentially a contiguous version of a histogram. We can see that by overlaying a KDE on top of the histogram of logs that we have created before:

```
# Create the base
base <- ggplot(db@data, aes(x=logpr))
# Histogram
hist <- base +
  geom_histogram(bins=50, aes(y=..density..))
# Overlay density plot
kde <- hist +
```

```
  geom_density(fill="#FF6666", alpha=0.5, colour="#FF6666")
# Change the background to go with the HTML color
final <- kde +
  theme(plot.background = element_rect(fill = '#fffff8'))
final
```

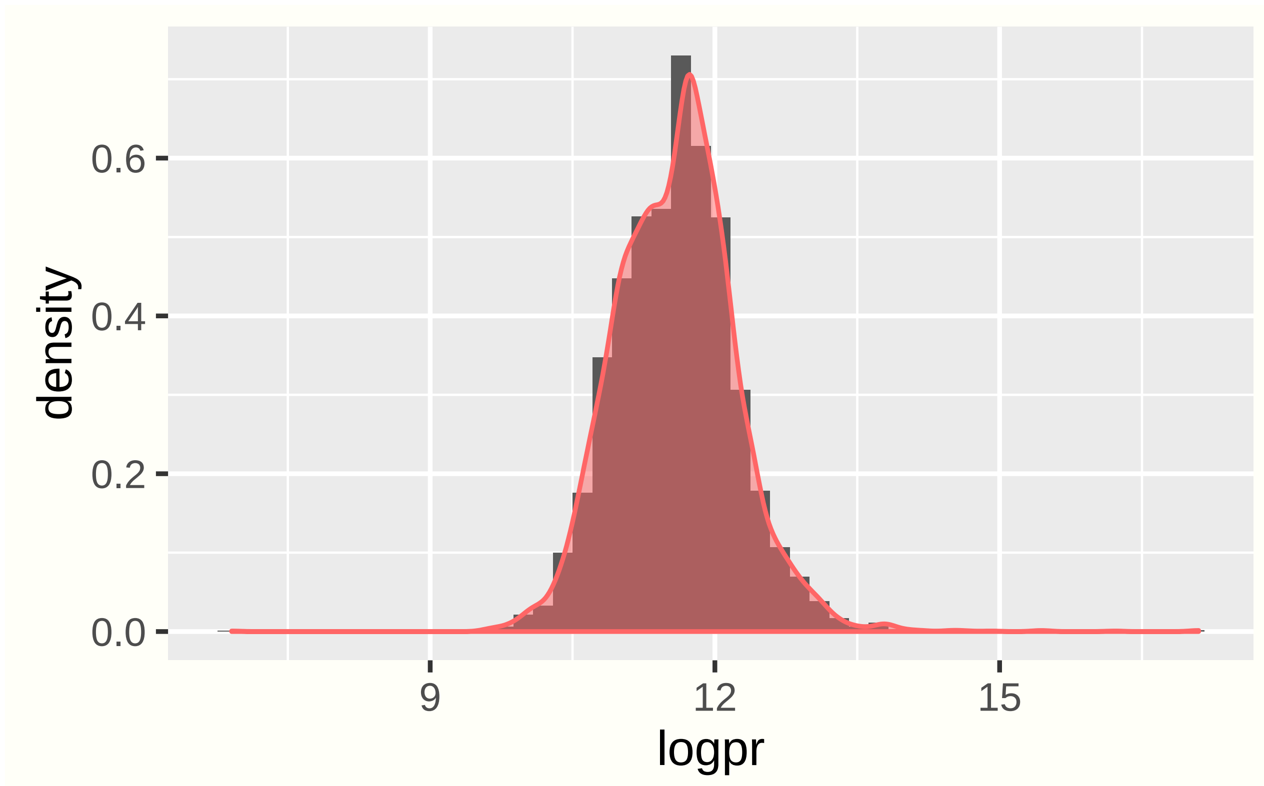


Figure 4: Histogram and KDE of the log of house prices in Liverpool

The key idea is that we are smoothing out the discrete binning that the histogram involves. Note how the histogram is exactly the same as above shape-wise, but it has been rescalend on the Y axis to reflect probabilities rather than counts.

Two-dimensional (spatial) KDE

Geography, at the end of the day, is usually represented as a two-dimensional space where we locate objects using a system of dual coordinates, `X` and `Y` (or latitude and longitude). Thanks to that, we can use the same technique as above to obtain a smooth representation of the distribution of a two-dimensional variable. The crucial difference is that, instead of obtaining a curve as the output, we will create a *surface*, where intensity will be represented with a color gradient, rather than with the second dimension, as it is the case in the figure above.

To create a spatial KDE in R, there are several ways. If you do not want to necessarily acknowledge the spatial nature of your data, or you they are not stored in a spatial format, you can plot them using `ggplot2`. Note we first need to convert the coordinates (stored in the spatial part of `db`) into columns of X and Y coordinates, then we can plot them:

```
# Attach XY coordinates
db@data['X'] <- db@coords[, 1]
db@data['Y'] <- db@coords[, 2]
# Set up base layer
base <- ggplot(data=db@data, aes(x=X, y=Y))
# Create the KDE surface
kde <- base + stat_density2d(aes(x = X, y = Y, alpha = ..level..),
               size = 0.01, bins = 16, geom = 'polygon') +
            scale_fill_gradient()
# Change the background to go with the HTML color
final <- kde +
  theme(plot.background = element_rect(fill = '#fffff8'))
final
```

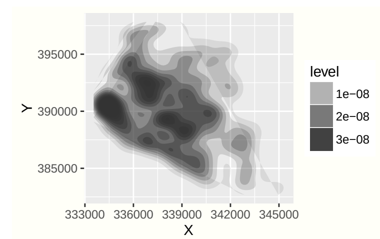


Figure 5: KDE of house transactions in L

Or, we can use a package such as the `GISTools`, which allows to pass a spatial object directly:

```
# Compute the KDE
kde <- kde.points(db)
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared fi
```

```
# Plot the KDE
level.plot(kde)
```

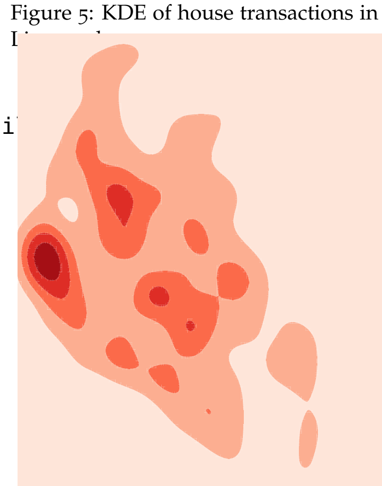


Figure 6: KDE of house transactions in Liverpool

Either of these approaches generate a surface that represents the density of dots, that is an estimation of the probability of finding a house transaction at a given coordinate. However, without any further information, they are hard to interpret and link with previous knowledge of the area. To bring such context to the figure, we can plot an underlying basemap, using a cloud provider such as Google Maps or, as in this case, OpenStreetMap. To do it, we will leverage the library `ggmap`, which is designed to play nicely with the `ggplot2` family (hence the seemingly counterintuitive example above). Before we can plot them with the online map, we need to reproject them though.

```
# Reproject coordinates
wgs84 <- CRS("+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0")
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
```

```
db_wgs84 <- spTransform(db, wgs84)
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
```

```
db_wgs84@data['lon'] <- db_wgs84@coords[, 1]
db_wgs84@data['lat'] <- db_wgs84@coords[, 2]
xys <- db_wgs84@data[c('lon', 'lat')]
# Bounding box
liv <- c(left = min(xys$lon), bottom = min(xys$lat),
         right = max(xys$lon), top = max(xys$lat))
# Download map tiles
basemap <- get_stamenmap(liv, zoom = 12,
                         maptype = "toner-lite")
# Overlay KDE
final <- ggmap(basemap, extent = "panel",
               maprange=FALSE) +
  stat_density2d(data = db_wgs84@data,
                 aes(x = lon, y = lat,
                     alpha=..level..,
                     fill = ..level..),
                 size = 0.01, bins = 16,
                 geom = 'polygon',
                 show.legend = FALSE) +
  scale_fill_gradient2("Transaction\nDensity",
                       low = "#fffff8",
                       high = "#8da0cb")
# Change the background to go with the HTML color
final <- final +
  theme(plot.background = element_rect(fill = '#fffff8'),
        legend.background = element_rect(fill = '#fffff8'),
        plot.background = element_rect(fill = '#fffff8'))
final
```

The plot above[3] allows us to not only see the distribution of house transactions, but to relate it to what we know about Liverpool, allowing us to establish many more connections than we were previously able. Mainly, we can easily see that the area with a highest volume of houses being sold is the city centre, with a "hole" around it that displays very few to no transactions and then several pockets further away.

[3] **EXERCISE** The map above uses the Stamen map `toner-lite`. Explore additional tile styles on their website and try to recreate the plot above.

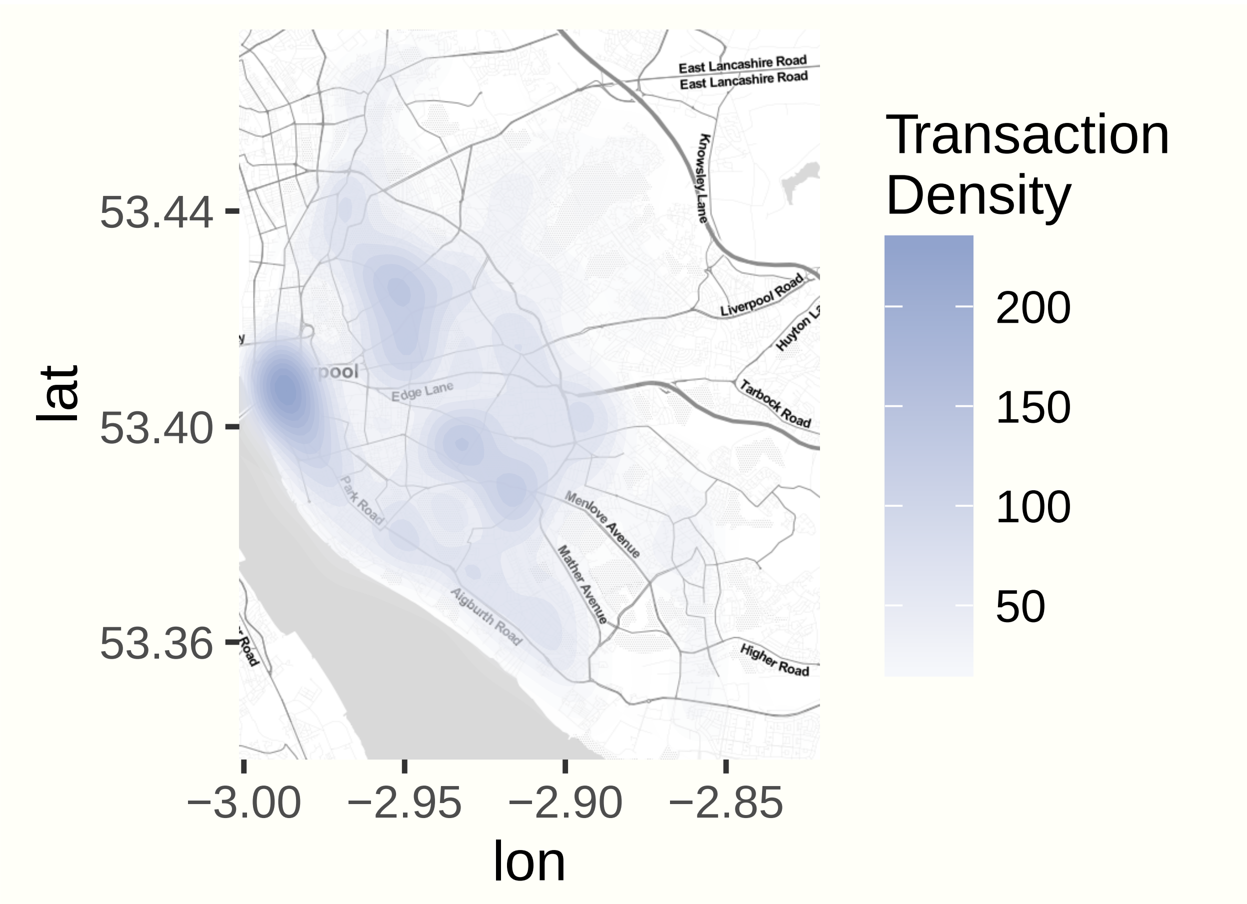


Figure 7: KDE of house transactions in Liverpool

Spatial Interpolation

The previous section demonstrates how to visualize the distribution of a set of spatial objects represented as points. In particular, given a bunch of house transactions, it shows how one can effectively visualize their distribution over space and get a sense of the density of occurrences. Such visualization, because it is based on KDE, is based on a smooth continuum, rather than on a discrete approach (as a choropleth would do, for example).

Many times however, we are not particularly interested in learning about the density of occurrences, but about the distribution of a given value attached to each location. Think for example of weather stations and temperature: the location of the stations is no secret and rarely changes, so it is not of particular interest to visualize the density of stations; what we are usually interested instead is to know how temperature is distributed over space, given we only measure it in a few places. One could argue the example we have been working with so far, house price transactions, fits into this category as well: although where house are sold may be of relevance, more often we are interested in finding out what the "surface of price" looks like. Rather than *where are most houses being sold?* we usually want to know *where the most expensive or most affordable* houses are located.

In cases where we are interested in creating a surface of a given value, rather than a simple density surface of occurrences, KDE cannot help us. In these cases, what we are interested in is *spatial interpolation*, a family of techniques that aim at exactly that: creating continuous surfaces for a particular phenomenon (e.g. temperature, house prices) given only a finite sample of observations. Spatial interpolation is a large field of research that is still being actively developed and that can involve a substantial amount of mathematical complexity in order to obtain the most accurate estimates possible[4]. In this session, we will introduce the simplest possible way of interpolating values, hoping this will give you a general understanding of the methodology and, if you are interested, you can check out further literature. For example, Banerjee, Carlin, and Gelfand (2014) or Cressie (2015) are hard but good overviews.

[4] There is also an important economic incentive to do this: some of the most popular applications are in the oil and gas or mining industries. In fact, the very creator of this technique, Danie G. Krige, was a mining engineer. His name is usually used to nickname spatial interpolation as *kriging*.

Inverse Distance Weight (IDW) interpolation

The technique we will cover here is called *Inverse Distance Weighting*, or IDW for convenience. Brunsdon and Comber (2015) offer a good description:

> In the *inverse distance weighting* (IDW) approach to interpolation, to estimate the value of z at location x a weighted mean of nearby obser-

> vations is taken [...]. To accommodate the idea that observations of z at points closer to x should be given more importance in the interpolation, greater weight is given to these points [...]
>
> — Page 204

The math[5] is not particularly complicated and may be found in detail elsewhere (the reference above is a good starting point), so we will not spend too much time on it. More relevant in this context is the intuition behind. Essentially, the idea is that we will create a surface of house price by smoothing many values arranged along a regular grid and obtained by interpolating from the known locations to the regular grid locations. This will give us full and equal coverage to soundly perform the smoothing.

[5] Essentially, for any point x in space, the IDW estimate for value z is equivalent to $\hat{z}(x) = \frac{\sum_i w_i z_i}{\sum_i w_i}$ where i are the observations for which we do have a value, and w_i is a weight given to location i based on its distance to x.

Enough chat, let's code.

From what we have just mentioned, there are a few steps to perform an IDW spatial interpolation:

1. Create a regular grid over the area where we have house transactions.
2. Obtain IDW estimates for each point in the grid, based on the values of k nearest neighbors.
3. Plot a smoothed version of the grid, effectively representing the surface of house prices.

Let us go in detail into each of them[6]. First, let us set up a grid:

[6] For the relevant calculations, we will be using the `gstat` library.

```
liv.grid <- spsample(db, type='regular', n=25000)
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
```

That's it, we're done! The function `spsample` hugely simplifies the task by taking a spatial object and returning the grid we need. Not a couple of additional arguments we pass: `type` allows us to get a set of points that are *uniformly* distributed over space, which is handy for the later smoothing; `n` controls how many points we want to create in that grid.

On to the IDW. Again, this is hugely simplified by `gstat`:

```
idw.hp <- idw(price ~ 1, locations=db, newdata=liv.grid)
```

```
## [inverse distance weighted interpolation]
```

```
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
```

Boom! We've got it. Let us pause for a second to see how we just did it. First, we pass `price ~ 1`. This specifies the formula we are using to model house prices. The name on the left of ~ represents the variable we want to explain, while everything to its right captures the *explanatory* variables. Since we are considering the simplest possible case, we do not have further variables to add, so we simply write `1`. Then we specify the original locations for which we do have house prices (our original `db` object), and the points where we want to interpolate the house prices (the `liv.grid` object we just created above). One more note: by default, `idw.hp` uses all the available observations, weighted by distance, to provide an estimate for a given point. If you want to modify that and restrict the maximum number of neighbors to consider, you need to tweak the argument `nmax`.

The object we get from `idw` is another spatial table, just as `db`, containing the interpolated values. As such, we can inspect it just as with any other of its kind. For example, to check out the top of the estimated table:

```
head(idw.hp@data)
```

```
##   var1.pred var1.var
## 1  158099.8       NA
## 2  158210.7       NA
## 3  158324.4       NA
## 4  158441.0       NA
## 5  158560.7       NA
## 6  158683.5       NA
```

The column we will pay attention to is `var1.pred`. And to see the locations for which those correspond:

```
head(idw.hp@coords)
```

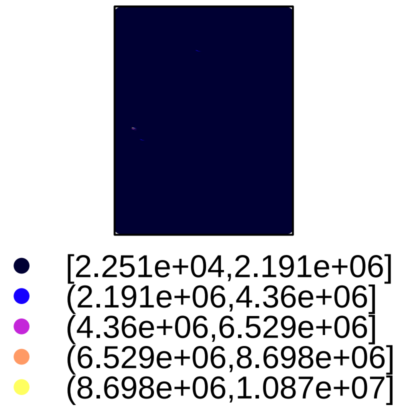
```
##            x1       x2
## [1,] 333599.4 382706.3
## [2,] 333684.3 382706.3
## [3,] 333769.3 382706.3
## [4,] 333854.2 382706.3
## [5,] 333939.2 382706.3
## [6,] 334024.2 382706.3
```

So, for a hypothetical house sold at the location in the first row of `idw.hp@coords` (expressed in the OSGB coordinate system), the price we would guess it would cost, based on the price of houses sold nearby, is the first element of column `var1.pred` in `idw.hp@data`.

A surface of housing prices

Once we have the IDW object computed, we can plot it to explore the distribution, not of house transactions in this case, but of house price over the geography of Liverpool. The easiest way to do this is by quickly calling the command `spplot`:

```
spplot(idw.hp['var1.pred'])
```



However, this is not entirely satisfactory for a number of reasons. First, it is not really a surface, but a quick representation of the points we have just estimated. To solve this, we need to do an interim transformation to convert the spatial object `idw.hp` into a table that a "surface plotter" can understand.

```
xyz <- data.frame(x=coordinates(idw.hp)[, 1],
                  y=coordinates(idw.hp)[, 2],
                  z=idw.hp$var1.pred)
```

Now we are ready to plot the surface in all its glory:

```
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base + geom_contour(aes(z=z))
surface
```

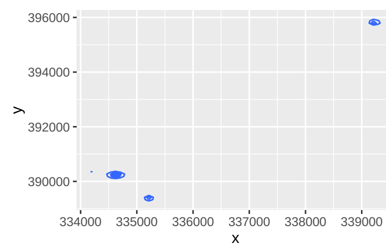


Figure 8: Contour of prices in Liverpool

This is a bit better but, arguably, not perfect yet. To build a more compelling visualization, we need to introduce a few more lines of code:

```
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base +
           geom_raster(aes(fill=z, alpha=z)) +
           scale_fill_gradient2("Transaction\nDensity",
                       low = "#fffff8",
                       high = "#9b59b6") +
          geom_contour(aes(z=z), colour='#9b59b6')
surface
```

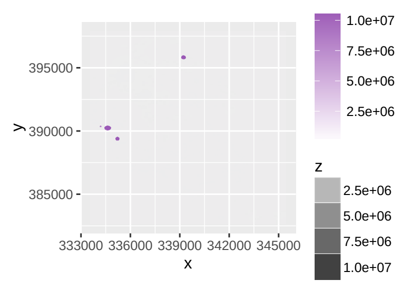


Figure 9: Surface of prices in Liverpool

The problem here, when compared to the KDE above for example, is that a few values are extremely large:

```
qplot(data=xyz, x=z, geom='density')
```

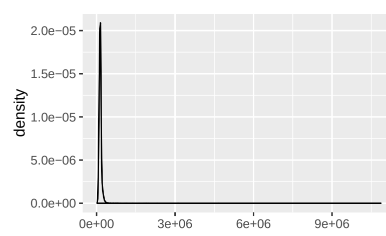


Figure 10: Skewness of prices in Liverpool

Let us then take the logarithm before we plot the surface:

```
xyz['lz'] <- log(xyz$z)
base <- ggplot(data=xyz, aes(x=x, y=y))
surface <- base +
           geom_raster(aes(fill=lz, alpha=lz),
                       show.legend = FALSE) +
           scale_fill_gradient2(low = "#fffff8",
                       high = "#66c2a5") +
```

```
            theme(legend.position = "none")
final <- surface +
  theme(plot.background = element_rect(fill = '#fffff8'),
        legend.background = element_rect(fill = '#fffff8'),
        plot.background = element_rect(fill = '#fffff8'))
final
```

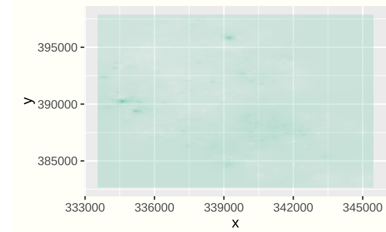


Figure 11: Surface of log-prices in Liverpool

Now this looks better. We can already start to tell some patterns. To bring in context, it would be great to be able to add a basemap layer, as we did for the KDE. This is conceptually very similar to what we did above, starting by reprojecting the points and continuing by overlaying them on top of the basemap. However, technically speak it is not possible because `ggmap` –the library we have been using to display tiles from cloud providers– does not play well with our own rasters (i.e. the price surface). For that reason, we will use an alternative basemap to provide context. In particular, we will overlay a shapefile that contains the outline of the local authority of Liverpool.

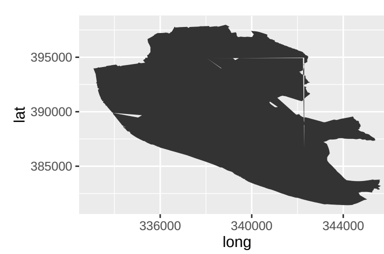
Let us first load up the new shapefile and process it to display it[7]:

[7] Instructions to process it are based on this tutorial by Hadley Wickham.

```
# Load up the layer
liv.otl <- readOGR('house_transactions', 'liv_outline')

## OGR data source with driver: ESRI Shapefile
## Source: "house_transactions", layer: "liv_outline"
## with 1 features
## It has 1 fields

## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files

# Assign the row names as identifiers
liv.otl@data$id <- rownames(liv.otl@data)
liv.otl.points <- fortify(liv.otl, region='id')

## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files
## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files

liv.otl.df <- join(liv.otl.points, liv.otl@data, by='id')
```

The shape we will overlay looks like this:

```
ggplot(liv.otl.df) +
  aes(long, lat) +
  geom_polygon()
```



Then, overlaying the surface of log-prices is relatively straightforward:

```
xyz['lz'] <- log(xyz$z)
base <- ggplot(liv.otl.df) +
            aes(long, lat) +
            geom_polygon()
surface <- base +
           geom_raster(data=xyz,
                       aes(x=x, y=y,
                           fill=lz, alpha=lz),
                           show.legend = FALSE) +
           scale_fill_gradient2(low = "#fffff8",
                       high = "#66c2a5") +
           theme(legend.position = "none")
final <- surface +
  theme(plot.background = element_rect(fill = '#fffff8'),
        panel.background = element_rect(fill = '#fffff8'),
        legend.background = element_rect(fill = '#fffff8'),
        plot.background = element_rect(fill = '#fffff8'))
final
```

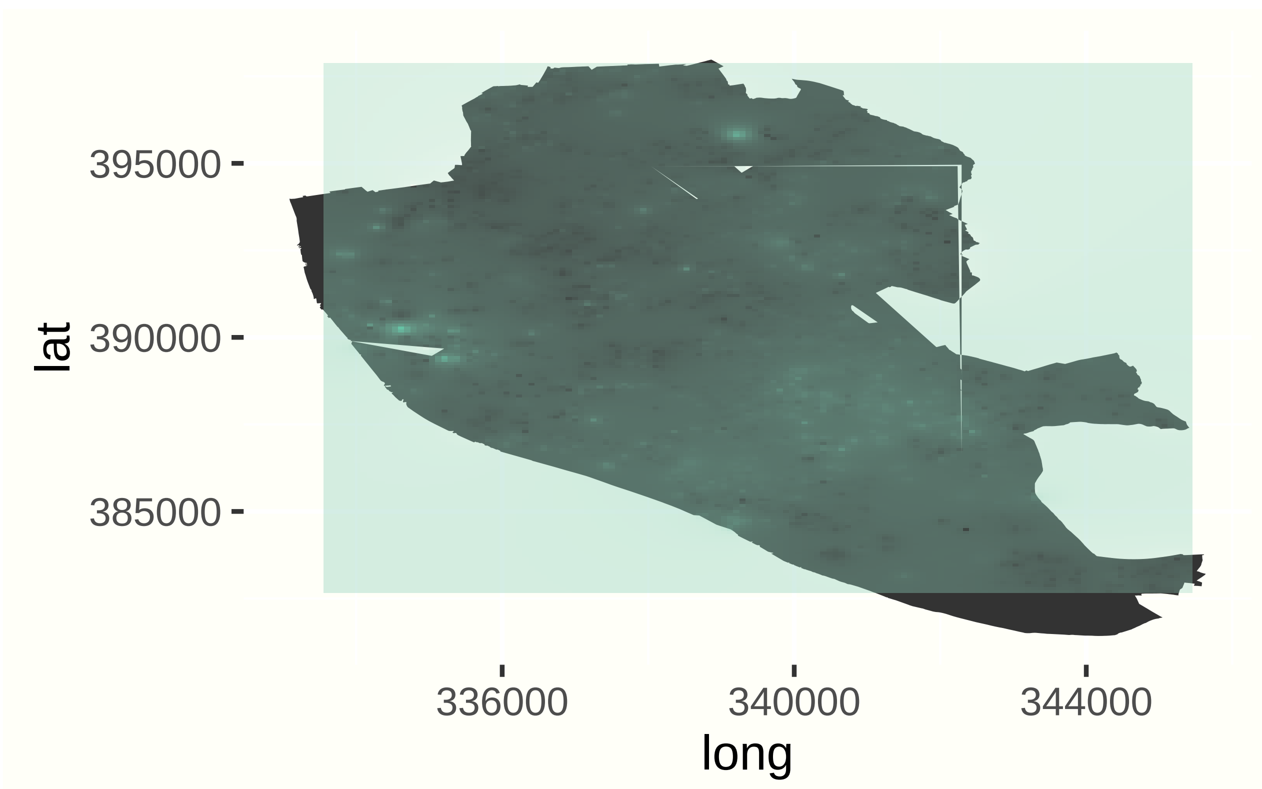


Figure 12: Surface of log-prices in Liverpool

"What should the next house's price be?"

The last bit we will explore in this session relates to prediction for new values. Imagine you are a real state data scientist and your boss asks you to give an estimate of how much a new house going into the market should cost. The only information you have to make such a guess is the location of the house. In this case, the IDW model we have just fitted can help you. The trick is realizing that, instead of creating an entire grid, all we need is to obtain an estimate of a single location.

Let us say, the house is located on the coordinates `x=34000`, `y=39000` as expressed in the GB National Grid coordinate system. In that case, we can do as follows:

```
pt <- SpatialPoints(cbind(x=34000, y=39000),
                    proj4string = db@proj4string)
idw.one <- idw(price ~ 1, locations=db, newdata=pt)

## [inverse distance weighted interpolation]

## NOTE: rgdal::checkCRSArgs: no proj_defs.dat in PROJ.4 shared files

idw.one

##       coordinates var1.pred var1.var
## 1 (34000, 39000)  144595.5       NA
```

And, as show above, the estimated value is GBP144,595.5[8].

[8] **PRO QUESTION** Is that house expensive or cheap, as compared to the other houses sold in this dataset? Can you figure out where the house is?

References

Banerjee, Sudipto, Bradley P Carlin, and Alan E Gelfand. 2014. *Hierarchical Modeling and Analysis for Spatial Data*. Crc Press.

Bivand, Roger S, Edzer Pebesma, and Virgilio Gómez-Rubio. 2013. *Applied Spatial Data Analysis with R*. Springer New York.

Brunsdon, Chris, and Lex Comber. 2015. *An Introduction to R for Spatial Analysis & Mapping*. Sage.

Cressie, Noel. 2015. *Statistics for Spatial Data*. John Wiley & Sons.

Lovelace, Robin, and James Cheshire. 2014. "Introduction to visualising spatial data in R." *National Centre for Research Methods Working Papers* 14 (03). `https://github.com/Robinlovelace/Creating-maps-in-R`.