Using huff-tools to estimate retail catchments with R

Introduction to catchment models and retail

There are several techniques of estimating potential retail catchment areas. Perhaps the simplest way is to draw buffer rings around a store or to use drive times or drive distances; however, such techniques are simplistic and are unlikely to sufficiently capture the complexity of different attributes that may influence true catchment extent. Another approach is to use more sophisticated methods such as spatial interaction models often referred to as gravity models. They apply Newtonian laws of physics to the modelling of shopper behaviour based on the relative attractiveness of different shopping destinations which decays with the distance to potential consumer domiciles. The map below shows catchment areas for Liverpool Central estimated by using buffers and drive distances

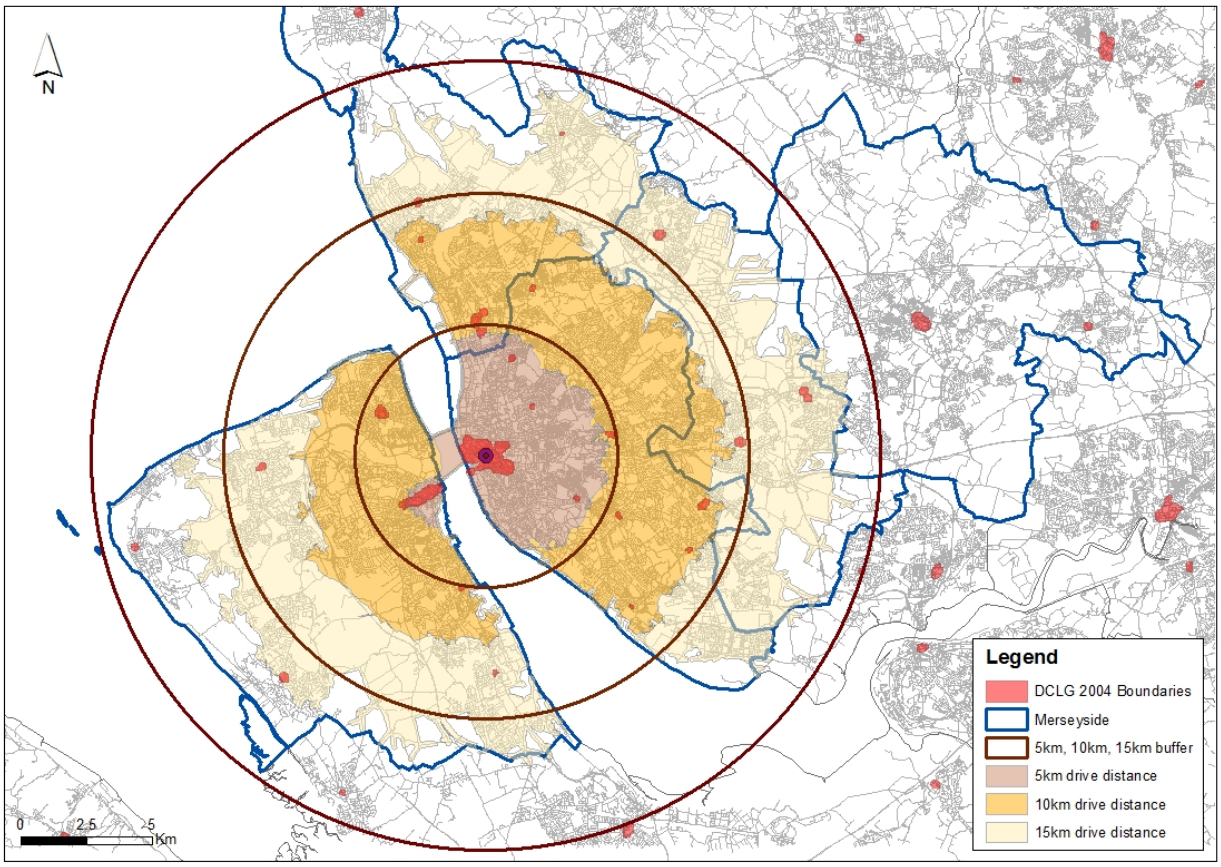


Figure 1: Buffer vs. drive distance

One of the most popular such methods is Huff model which estimates the probability for people to visit different shopping destinations. Most commonly, it is applied to a single store, chain of stores or shopping centres; therefore its application to an entire retail centre or a number of retail centres has some challenges. First, the potential catchment areas for different retail/service types (e.g. convenience, comparison retail) are likely to vary substantially as consumers are willing to travel further to purchase higher order comparison goods. When estimating catchment areas for entire retail centres a consideration needs to be given whether to use total retail/service numbers or a particular retail type only e.g. comparison. Second, the extent of catchments is also likely to be affected by the position of a centre within retail hierarchy capturing their competitive advantage and multidimensional functions. on etc.

What you should know

a. Basic knowlege of running and manipulating data using R.

b. Understand what an R function is.

c. Understand what a Shapefile is, and how it might be mapped in a GIS.

Overview

This tutorial covers the following topics:

1. Introduction to R.
2. Calculating the probability for people to visit different retail centres within the Leeds City Region.
3. Extracting catchment areas and exporting to a Shapefile.
4. Estimating the characteristics of population flows to retail centres.
5. Evaluating the effect of different parameter values.

The Huff model has 4 parameters that are used to estimate the probability of people visiting a particular retail centre, namely:

A measure of attractiveness of each retail centre

Most common measures of retail centre attractiveness are related to the size, with larger centres having been normally associated with higher attractiveness, due to more variety and better customer experience offered. Typically a gross/net selling area is employed in such a measure but also the number of retail outlets, often scoring multiples higher that the independent retailers has been used to proxy town centre attractiveness. It is important however, to note that single measures of attractiveness such as centre size are far from being comprehensive. There is empirical evidence that town centre attractiveness in the context of consumer's choices of shopping destination is also impacted by the presence of specific anchor stores, retail tenant mix and 'non-retail tenant mix' such as leisure outlets.
We have estimated attractiveness of retail centres by creating a composite Index of Attractiveness, based on town centre composition data, obtained from LDC. It accounts for the size of each town (proxied by a number of businesses as vacant units can be a deterrent to the perceived centre's attractiveness) giving additional weights to comparison retail, leisure units and the most attractive/anchor stores which tend to generate large footfall. The method used to create the composite index involves standardizing the four variables by applying a range from 0 to 100 and summing up the scores.

The alpha exponent of the attractiveness score

The alpha exponent is a parameter to which the attractiveness value is raised to enable modelling nonlinear behaviour of the attractiveness variable such as ease of access to a particular retail centre, perception of attractiveness, trading hours etc. (normally estimated from empirical observations). In our model we increase the attractiveness value based on ease of access to a particular centre, so if the distance is less than 500m, we are more likely to use this centre for a 'top up' shopping. As a result, we set alpha values based on distance, that is if distance is less than 0.5 Km then alpha = 2, in other cases alpha = 1.

The distance between each residential zone and the retail centre

There are two most common ways of calculating the distance between customers domicile and a town centre. First, the Euclidean straight line distance - this is a simplistic approach as it does not account for any logistical barriers. Second, the travel distance which uses existing OS road network - it provides more accurate distance estimates, therefore it was our preferred approach. The comparison between straight line distance and drive distance between Portsmouth and Fareham is shown on the map below.

The beta exponent of distance

Based on the Huff retail gravity model, distance is raised to the power of a beta exponent in order to model the negative relationship between distance and retail attractiveness. The beta exponent usually takes a value between -1 and -2. As can be seen from the following graph, the lower the value of beta the lower the attractiveness. Therefore, lower beta values are assigned to retail centres that we assume that the attratciveness (and probability of patronage) is reduced faster over distance.

For simplicity, this tutorial is based on pre-calculated input data for the limit extent of the Leeds City Region; however, the model is applicable within different contexts or for national extents. The pre-calculated input data for this tutorial comprises:

a. A shapefile of the Lower Layer Super Output Areas for the region of interest - in this case, the Leeds City Region (details to download and reproduce the data can be found here).

b. A shapefile of the boundaries of retail centres - in this case, these relate to the DCLG 2004 definitions.

c. A csv file of distances between the LSOA centroids and the retail centre boundaries (an example of calculating road distances with huff-tools is provided here).

d. A csv file of attractiveness scores for the retail centres (an example of producing these in R is provided here).

e. A csv file of the population for each LSOA from the 2011 Census (details to download and reproduce the data are provided here).

All five files can be downloaded from here.

1. Introduction to R

This tutorial uses the R programming language, and a short introduction to R is provided below. The huff_basic function is used to apply the Huff model which is part of the huff-tools package and for that reason we should import huff-tools in R. We also need to set the working directory to the local data directory. A detailed explanation of the available functions in huff-tools along with examples of using the library in R can be found here.

```
# Change working directory
setwd("~/Dropbox/liverpool/retail/PresentationLDC")
# Import the huff-tools library
source("huff-tools.r")
```

```
## Loading required package: sp
```

```
## rgdal: version: 1.1-10, (SVN revision 622)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
```

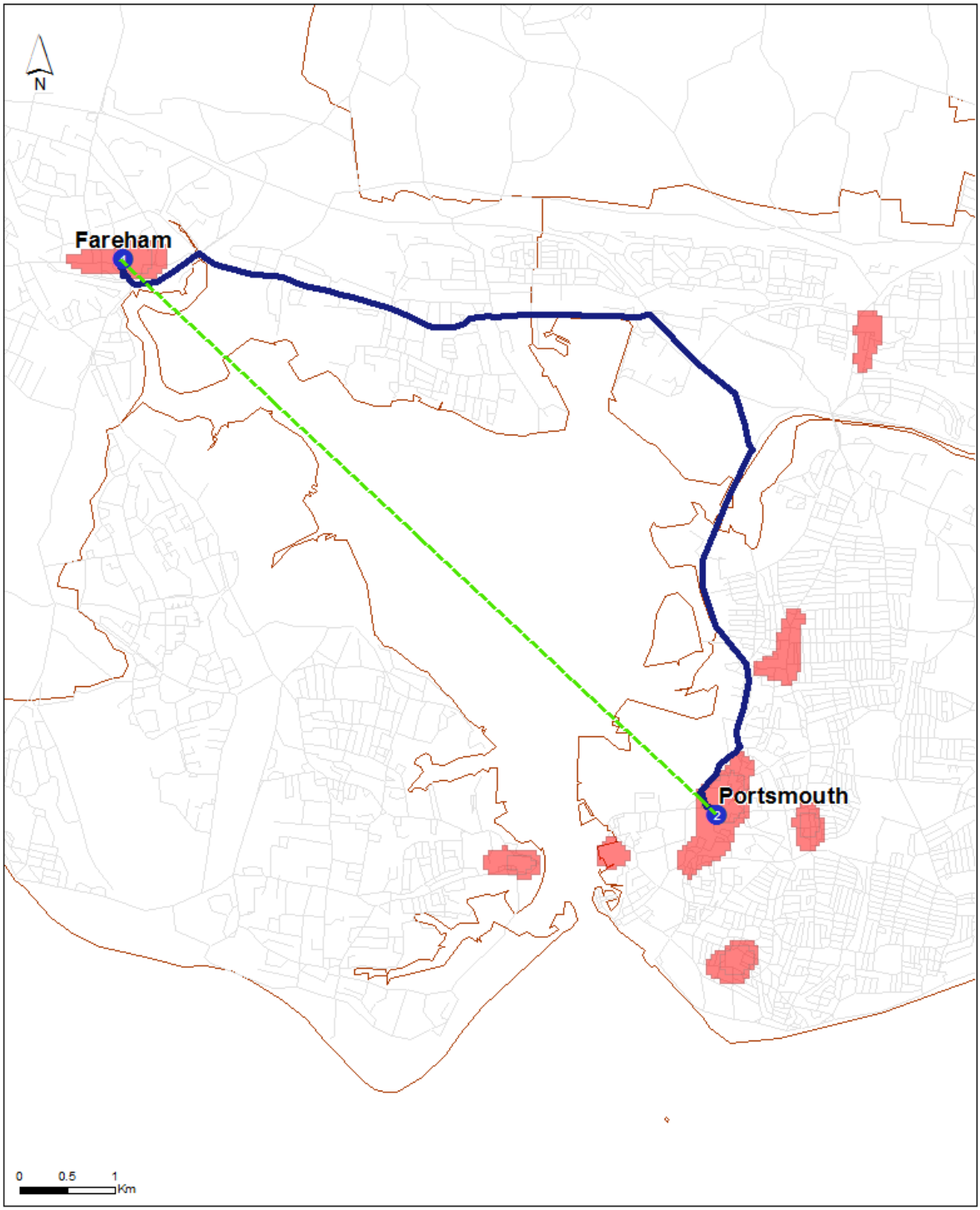


Figure 2: Straight line vs. drive distance

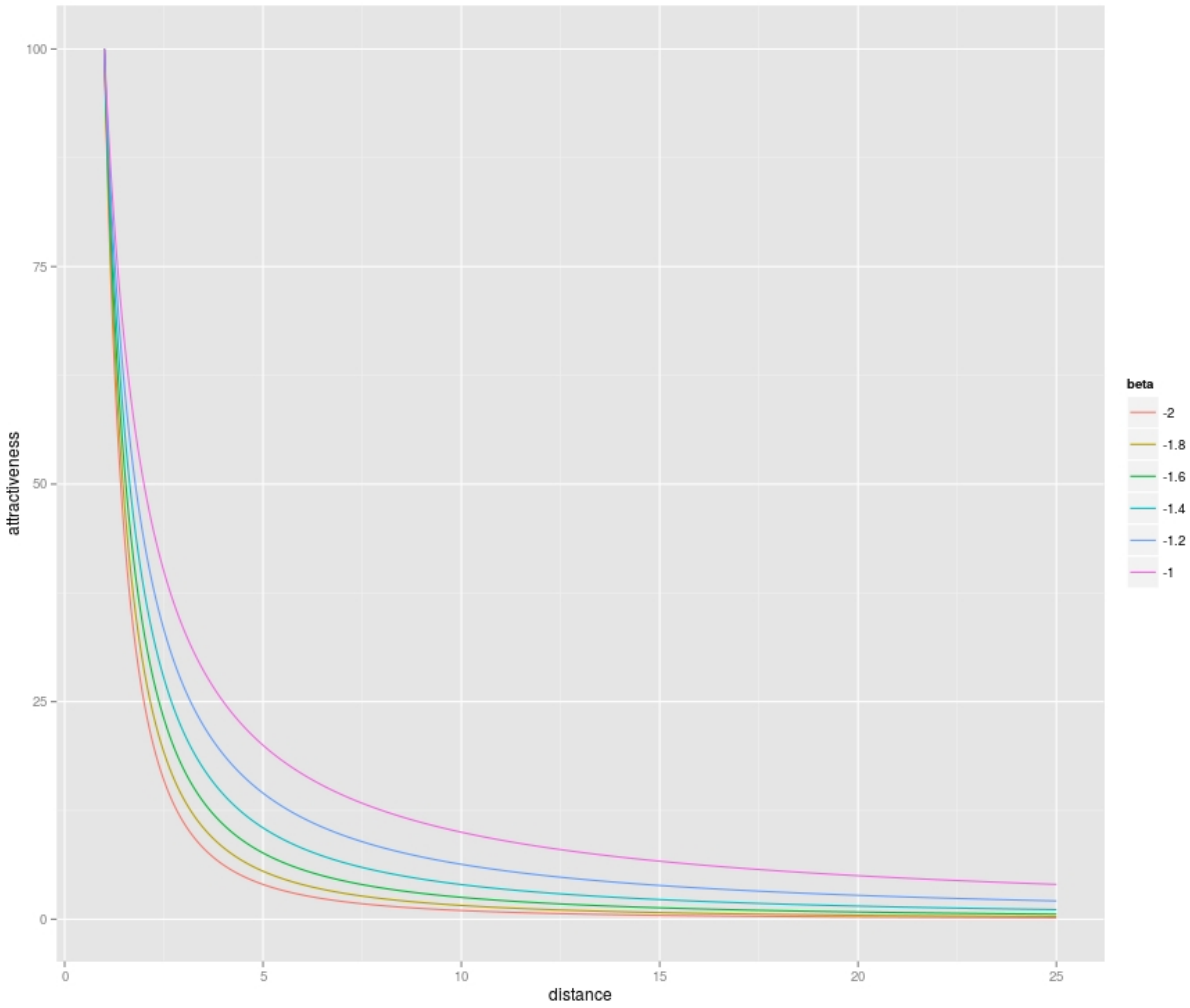


Figure 3: Distance decay parameter for different beta values

```
##  Loaded GDAL runtime: GDAL 1.11.3, released 2015/09/16
##  Path to GDAL shared files: /usr/share/gdal/1.11
##  Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
##  Path to PROJ.4 shared files: (autodetected)
##  Linking to sp version: 1.2-3

## rgeos version: 0.3-19, (SVN revision 524)
##  GEOS runtime version: 3.5.0-CAPI-1.9.0 r4084
##  Linking to sp version: 1.2-3
##  Polygon checking: TRUE

##
## Attaching package: 'igraph'

## The following object is masked from 'package:rgeos':
##
##     union

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union

##
## Attaching package: 'FNN'

## The following object is masked from 'package:igraph':
##
##     knn

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:igraph':
##
##     %>%, as_data_frame, groups, union

## The following objects are masked from 'package:rgeos':
##
##     intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Next, we will use the 'read.csv' function to import comma-delimited files into R. In most cases, the function can be applied by specifying just the name of the csv file as it assumes by default that the csv file has a header and that the field separator character is ','. These and other parameters can be modified if required (type ?read.csv for more details). The function returns a data frame object and can be used as follows:

```
setwd("~/Dropbox/liverpool/retail/PresentationLDC/data")
attr_score <- read.csv('LCR_attractiveness.csv')
distances <- read.csv('distances.csv')
```

We have now created two data frame objects. The "attr_score" data frame stores the attractivenes score of the retail centres (including ranking based on the hierarchy of the town centres) while the "distances" object has the pre-calculated road distances between the centroids of LSOAs and the boundary of the retail centres. The entries of a data frame can be accessed in various ways:

```
names(attr_score) # Column names
```

```
## [1] "Name"      "ID"        "Region"    "AttrScore" "Rank"
```

```
attr_score[, 1] # Get column by index
```

```
##  [1] Acomb                  Armley                 Barnsley
##  [4] Batley                 Bingley                Birstal Smithies
##  [7] Birstall Shopping Park Bradford               Brighouse
## [10] Castleford             Cleckheaton            Crossgates
## [13] Dewsbury               Garforth               Guiseley
## [16] Halifax                Halton, Leeds          Harehills
## [19] Harrogate              Headingley             Hebden Bridge
## [22] Heckmondwike           Holmfirth              Huddersfield
## [25] Ilkley                 Keighley               Knaresborough
## [28] Leeds                  Morley                 Normanton
## [31] Oakwood, Leeds         Ossett                 Otley
## [34] Pontefract             Pudsey                 Ripon
## [37] Selby                  Settle                 Shipley
## [40] Skipton                South Elmsall          Tadcaster
## [43] Todmorden              Wakefield              Wetherby
## [46] White Rose Centre      Wombwell               Yeadon
## [49] York
## 49 Levels: Acomb Armley Barnsley Batley Bingley ... York
```

```
attr_score$Name # Get column by name
```

```
##  [1] Acomb                  Armley                 Barnsley
##  [4] Batley                 Bingley                Birstal Smithies
##  [7] Birstall Shopping Park Bradford               Brighouse
## [10] Castleford             Cleckheaton            Crossgates
## [13] Dewsbury               Garforth               Guiseley
## [16] Halifax                Halton, Leeds          Harehills
## [19] Harrogate              Headingley             Hebden Bridge
## [22] Heckmondwike           Holmfirth              Huddersfield
## [25] Ilkley                 Keighley               Knaresborough
## [28] Leeds                  Morley                 Normanton
```

```
## [31] Oakwood, Leeds         Ossett                Otley
## [34] Pontefract             Pudsey                Ripon
## [37] Selby                  Settle                Shipley
## [40] Skipton                South Elmsall         Tadcaster
## [43] Todmorden              Wakefield             Wetherby
## [46] White Rose Centre      Wombwell              Yeadon
## [49] York
## 49 Levels: Acomb Armley Barnsley Batley Bingley ... York
```

```
attr_score[1, ] # Get row by index
```

```
##    Name     ID                    Region AttrScore Rank
## 1 Acomb TC0010 Yorkshire and the Humber      14.7    4
```

A summary output of the data can be obtained with the 'str' and 'summary' functions. The former function outputs the name and type of each variable as well as the number of rows, while the latter function produces summaries depending on the type of the variable (e.g. quartiles, minimum and maximum values for continuous variables).

```
str(attr_score)
```

```
## 'data.frame':    49 obs. of  5 variables:
##  $ Name     : Factor w/ 49 levels "Acomb","Armley",..: 1 2 3 4 5 7 6 8 9 10 ...
##  $ ID       : Factor w/ 49 levels "TC0010","TC0033",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ Region   : Factor w/ 1 level "Yorkshire and the Humber": 1 1 1 1 1 1 1 1 1 1 ...
##  $ AttrScore: num  14.7 11.7 75.8 19.2 14 7.1 33.9 83.7 36.7 39.9 ...
##  $ Rank     : int  4 4 2 4 4 4 3 2 3 3 ...
```

```
summary(attr_score)
```

```
##                      Name          ID                           Region
##  Acomb                 : 1   TC0010 : 1   Yorkshire and the Humber:49
##  Armley                : 1   TC0033 : 1
##  Barnsley              : 1   TC0076 : 1
##  Batley                : 1   TC0086 : 1
##  Bingley               : 1   TC0121 : 1
##  Birstall Shopping Park: 1   TC0126 : 1
##  (Other)               :43   (Other):43
##    AttrScore           Rank
##  Min.   :  5.00   Min.   :1.000
##  1st Qu.: 13.90   1st Qu.:3.000
##  Median : 24.60   Median :3.000
##  Mean   : 43.21   Mean   :3.082
##  3rd Qu.: 42.90   3rd Qu.:4.000
##  Max.   :300.00   Max.   :4.000
##
```

```
str(distances)
```

```
## 'data.frame':    90209 obs. of  3 variables:
##  $ ID         : Factor w/ 49 levels "TC0010","TC0033",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ origins_name: Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ distance   : num  58.6 58.8 58.5 58.5 58.9 ...
```

```
summary(distances)
```

```
##        ID           origins_name      distance
##  TC0010 : 1841   E01007317:   49   Min.   :  0.00
##  TC0033 : 1841   E01007318:   49   1st Qu.: 17.19
##  TC0076 : 1841   E01007319:   49   Median : 27.74
##  TC0086 : 1841   E01007320:   49   Mean   : 30.12
##  TC0121 : 1841   E01007321:   49   3rd Qu.: 40.53
##  TC0126 : 1841   E01007322:   49   Max.   :118.83
##  (Other):79163   (Other)  :89915
```

More advanced summary statistics can be produced with the 'aggregate' function that can split our data into subsets and then apply a function on each. For example, we can obtain the mean attractiveness score of the retail centres in our study area conditional on their hierarchy as follows:

```
aggregate(attr_score$AttrScore, by = list(attr_score$Rank), mean)
```

```
##   Group.1         x
## 1       1 179.90000
## 2       2  65.27143
## 3       3  31.53125
## 4       4  12.21429
```

The first argument of the function is the variable for which we want to produce summary statistics, the second argument provides the grouping elements and the third argument is the function to apply to each subset.

We can also import spatial data into R with the 'readOGR' function. The function accepts as first argument the directory where the vector dataset is located and as second argument the name of the dataset - note, there is no need to add a file extension. Running the following R snippet, we create an object of class SpatialPolygonsDataFrame.

```
destinations <- readOGR("/home/michalis/Dropbox/liverpool/retail/PresentationLDC/data",
                        "LCR_destinations")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/michalis/Dropbox/liverpool/retail/PresentationLDC/data", layer: "LCR_destinations"
## with 49 features
## It has 3 fields
```

A SpatialPolygonsDataFrame contains "slots" which store a matrix with the coordinates of the bounding box of the spatial object, a CRS object defining the coordinate reference system, attribute data and the spatial entities (i.e. polygons). We can access the slots of the SpatialPolygonsDataFrame with the '@' symbol. For example, to view the bounding box and CRS:

```
destinations@bbox
```

```
##      min    max
## x 381800 461650
## y 402900 471650
```

```
destinations@proj4string
```

```
## CRS arguments:
##  +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
```

We can also access the underlying spatial entities (i.e. polygons) via the 'polygons' slot, which has its own nested slots per polygon as can be seen below.

```
length(destinations@polygons) # The number of retail centres
```

```
## [1] 49
```

```
destinations@polygons[[1]]@area # The area of the first retail centre (m2)
```

```
## [1] 162500
```

Other useful information we often want to access are the attributes of the spatial entities. These can be accessed via the dedicated slot 'data' (which returns a data frame) in a similar manner as above:

```
str(destinations@data)
```

```
## 'data.frame':    49 obs. of  3 variables:
##  $ ID  : Factor w/ 49 levels "TC0010","TC0033",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ NAME: Factor w/ 49 levels "Acomb","Armley",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ AREA: num  16.2 7.5 9.5 61.5 23 ...
```

```
summary(destinations@data)
```

```
##        ID                          NAME         AREA
##  TC0010 : 1   Acomb                  : 1   Min.   :  2.25
##  TC0033 : 1   Armley                 : 1   1st Qu.:  7.50
##  TC0076 : 1   Barnsley               : 1   Median : 11.25
##  TC0086 : 1   Batley                 : 1   Mean   : 25.35
##  TC0121 : 1   Bingley                : 1   3rd Qu.: 22.75
##  TC0126 : 1   Birstall Shopping Park: 1   Max.   :223.75
##  (Other):43   (Other)                :43
```

It is often required to merge the information stored in a data frame with the attributes of a spatial object. This can be accomplished with the 'match' function as follows:

```
# verify that all retail centres have an attractiveness score
all(destinations@data$ID %in% attr_score$ID)
```

```
## [1] TRUE
```

```
# merge the two datasets
destinations@data <- data.frame(destinations@data,
                                attr_score[match(destinations@data$ID,
                                                 attr_score$ID), ])
# examine the result
str(destinations@data)
```

```
## 'data.frame':    49 obs. of  8 variables:
##  $ ID       : Factor w/ 49 levels "TC0010","TC0033",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ NAME     : Factor w/ 49 levels "Acomb","Armley",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ AREA     : num  16.2 7.5 9.5 61.5 23 ...
##  $ Name     : Factor w/ 49 levels "Acomb","Armley",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ ID.1     : Factor w/ 49 levels "TC0010","TC0033",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ Region   : Factor w/ 1 level "Yorkshire and the Humber": 1 1 1 1 1 1 1 1 1 1 ...
##  $ AttrScore: num  35.2 16.1 24.6 136.1 55.5 ...
##  $ Rank     : int  3 4 3 1 2 4 1 3 3 2 ...
```

The side-effect of this is that both fields that we merged on are now stored in the data frame of the spatial object, but it is easy to delete a field by setting it as NULL.

```
destinations@data$ID.1 <- NULL
str(destinations@data)
```

```
## 'data.frame':    49 obs. of  7 variables:
##  $ ID       : Factor w/ 49 levels "TC0010","TC0033",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ NAME     : Factor w/ 49 levels "Acomb","Armley",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ AREA     : num  16.2 7.5 9.5 61.5 23 ...
##  $ Name     : Factor w/ 49 levels "Acomb","Armley",..: 36 38 27 19 40 1 49 45 25 33 ...
##  $ Region   : Factor w/ 1 level "Yorkshire and the Humber": 1 1 1 1 1 1 1 1 1 1 ...
##  $ AttrScore: num  35.2 16.1 24.6 136.1 55.5 ...
##  $ Rank     : int  3 4 3 1 2 4 1 3 3 2 ...
```

Similarly, we might want to join two different data frames based on a common field, and given that the data frames 'distances' and 'attr_score' use the same name for the field that we want to merge on ('ID'), we can use the 'inner_join' function as follows:

```
huff_input <- inner_join(distances, attr_score) # Use the 'by' argument to specify column names if they
```

```
## Joining, by = "ID"
```

```
str(huff_input)
```

```
## 'data.frame':    90209 obs. of  7 variables:
##  $ ID          : Factor w/ 49 levels "TC0010","TC0033",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ origins_name: Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ distance    : num  58.6 58.8 58.5 58.5 58.9 ...
##  $ Name        : Factor w/ 49 levels "Acomb","Armley",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Region      : Factor w/ 1 level "Yorkshire and the Humber": 1 1 1 1 1 1 1 1 1 1 ...
##  $ AttrScore   : num  14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 ...
##  $ Rank        : int  4 4 4 4 4 4 4 4 4 4 ...
```

Another useful function that we will use to develop some of the parameters of the Huff model is the 'ifelse' function. This function returns an object which is filled with elements according to whether or not a condition is met. It is also possible to nest two or more 'ifelse' statements to evaluate more complex conditions.

For example, in this tutorial we use the retail centre hierarchy to develop the beta parameter (distance decay exponent) of the Huff model. Assuming that retail centres at the top of the hierarchy should have a lower distance decay parameter (i.e. their attractiveness is reduced at a slower rate with distance).

For the major retail centres, which serve extensive catchments (Rank 1) we will use beta = 1.4, for the secondary retail centres, typically sub-regional centres or major district centres (Rank 2) beta = 1.6. Lower retail hierarchy is assigned to small market towns and district centres (Rank 3) for which we select beta = 1.8, and finally for the smallest local centres and shopping parades (Rank 4) we use beta = 2.0.

```
# Using the 'with' function we don't have to use the full huff_input$Rank notation
huff_input$beta <- with(huff_input, ifelse(Rank == 1, 1.4,
                                    ifelse(Rank == 2, 1.6,
                                    ifelse(Rank == 3, 1.8, 2))))
# Display the first six values for Rank and beta fields
head(huff_input[,c("Rank", "beta")])
```

```
##   Rank beta
## 1    4    2
## 2    4    2
## 3    4    2
## 4    4    2
## 5    4    2
## 6    4    2
```

```
# Display the last six values for Rank and beta fields
tail(huff_input[,c("Rank", "beta")])
```

```
##       Rank beta
## 90204    1  1.4
## 90205    1  1.4
## 90206    1  1.4
## 90207    1  1.4
## 90208    1  1.4
## 90209    1  1.4
```

2. Calculate the Huff probabilities

2.1. Prepare input data

We will use the 'huff_basic' function to apply the Huff model. The 'huff_basic' function requires the input of 6 arguments to run, namely:

a. A vector of unique names for the destination locations.

b. A vector of score of attractiveness for the destination locations.

c. A vector of unique names for the origin locations.

d. A vector of pairwise distances between origins and destinations.

e. A vector or scalar for the alpha exponent of the attractiveness score.

f. A vector or scalar for the beta exponent of distance.

The first four arguments are required, while the last two are optional, and if not provided a default value will be used (i.e. alpha = 1, beta = 2). If you examine the huff_input data frame that we created earlier, we already have 5 of the variables that we need (i.e. names for destinations and origins, attractivenes score, distance and beta values).

```
str(huff_input)
```

```
## 'data.frame':    90209 obs. of  8 variables:
##  $ ID          : Factor w/ 49 levels "TC0010","TC0033",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ origins_name: Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ distance    : num  58.6 58.8 58.5 58.5 58.9 ...
##  $ Name        : Factor w/ 49 levels "Acomb","Armley",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Region      : Factor w/ 1 level "Yorkshire and the Humber": 1 1 1 1 1 1 1 1 1 1 ...
##  $ AttrScore   : num  14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 14.7 ...
##  $ Rank        : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ beta        : num  2 2 2 2 2 2 2 2 2 2 ...
```

So we only need to develop the alpha exponent. In our analysis we wanted to model a likely shopping behaviour that favours the nearest retail centre due to ease of access, so if the distance is less than 500m, we are more likely to use this centre for let's say 'top up' shopping. As a result, we set alpha values based on distance, that is if distance is less than 0.5 Km then alpha = 2 else alpha = 1.

```
huff_input$alpha <- with(huff_input, ifelse(distance <= 0.5, 2, 1))
```

It should be noted that both beta and alpha values can be updated, depending on the requirements. The above used values that were found to produce the most appropriate catchment areas for the national level. These may vary at the regional level for a number of reasons: a) the number of competitors is smaller within a region b) there is a violation of a boundary free modelling, and c) the regional hierarchy within retail centres may vary significantly from the national one.

2.2. Use the huff_basic function from huff-tools

To calculate the Huff probabilities we can now run the huff_basic function.

```
huff_probs <- huff_basic(huff_input$ID, huff_input$AttrScore, huff_input$origins_name,
                         huff_input$distance, huff_input$alpha, huff_input$beta)
```

```
## Warning in check_huff(destinations_name, destinations_attractiveness,
## origins_name, : Distances equal to zero were found, all distances below 0.1
## were replaced with 0.1
```

```
## Joining, by = "origins_name"
```

```
str(huff_probs)
```

```
## 'data.frame':    90209 obs. of  6 variables:
##  $ origins_name     : Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ destinations_name: Factor w/ 49 levels "TC0010","TC0033",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ distance         : num  58.6 58.8 58.5 58.5 58.9 ...
##  $ huff             : num  0.00428 0.00425 0.0043 0.0043 0.00423 ...
##  $ sum_huff         : num  23.9 34.7 20.9 30.1 17.6 ...
##  $ huff_probability : num  0.000179 0.000122 0.000206 0.000143 0.000241 ...
```

```
head(huff_probs)
```

```
##   origins_name destinations_name distance        huff sum_huff
## 1    E01007317            TC0010 58.61245 0.004278953 23.91284
## 2    E01007318            TC0010 58.79161 0.004252914 34.73234
## 3    E01007319            TC0010 58.49529 0.004296112 20.88507
## 4    E01007320            TC0010 58.46148 0.004301081 30.09935
## 5    E01007321            TC0010 58.94050 0.004231455 17.56538
## 6    E01007322            TC0010 57.53253 0.004441098 31.65076
##   huff_probability
## 1     0.0001789396
## 2     0.0001224483
## 3     0.0002057026
## 4     0.0001428961
## 5     0.0002408975
## 6     0.0001403157
```

The output of the huff-basic function is a table in long format (all pairwise probabilities between origins and destinations).

3. Extract the highest Huff probabilities for each LSOA

To extract the highest probabilities for each LSOA we can use the 'select_by_probs' function. The function will essentially group the data by LSOA name, sort each group entries by Huff probabilities (from higher to lower) and then extract the top number of entries, where the number is specified as second argument in the function. So to extract the highest entry (number = 1) for each LSOA we can run the following R snippet:

```
# Extract probabilities
sele_probs <- select_by_probs(huff_probs, 1)
```

3.1. Output the result as shapefile

In order to display the extracted Huff probabilities, we will create a shapefile by merging our 'sele_probs' data frame with a shapefile of LSOAs provided for our study area. First we import the shapefile.

```
origins <- readOGR("/home/michalis/Dropbox/liverpool/retail/PresentationLDC/data",
                   "LCR_lsoas")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/home/michalis/Dropbox/liverpool/retail/PresentationLDC/data", layer: "LCR_lsoas"
## with 1841 features
## It has 1 fields
```

And then merge the spatial dataset with our data frame.

```
# Merge with spatial object of LSOAs
origins@data <- data.frame(origins@data,
                           sele_probs[match(origins@data$LSOA11CD, sele_probs$origins_name), ])
str(origins@data)
```

```
## 'data.frame':    1841 obs. of  5 variables:
##  $ LSOA11CD         : Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ origins_name     : Factor w/ 1841 levels "E01007317","E01007318",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ destinations_name: Factor w/ 49 levels "TC0010","TC0033",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ distance         : num  2.81 1.99 3.4 2.26 3.95 ...
##  $ huff_probability : num  0.608 0.728 0.512 0.685 0.48 ...
```

```
origins@data[,1] <- NULL
```

We can also save the data as shapefile in a directory with the the 'writeOGR' function.

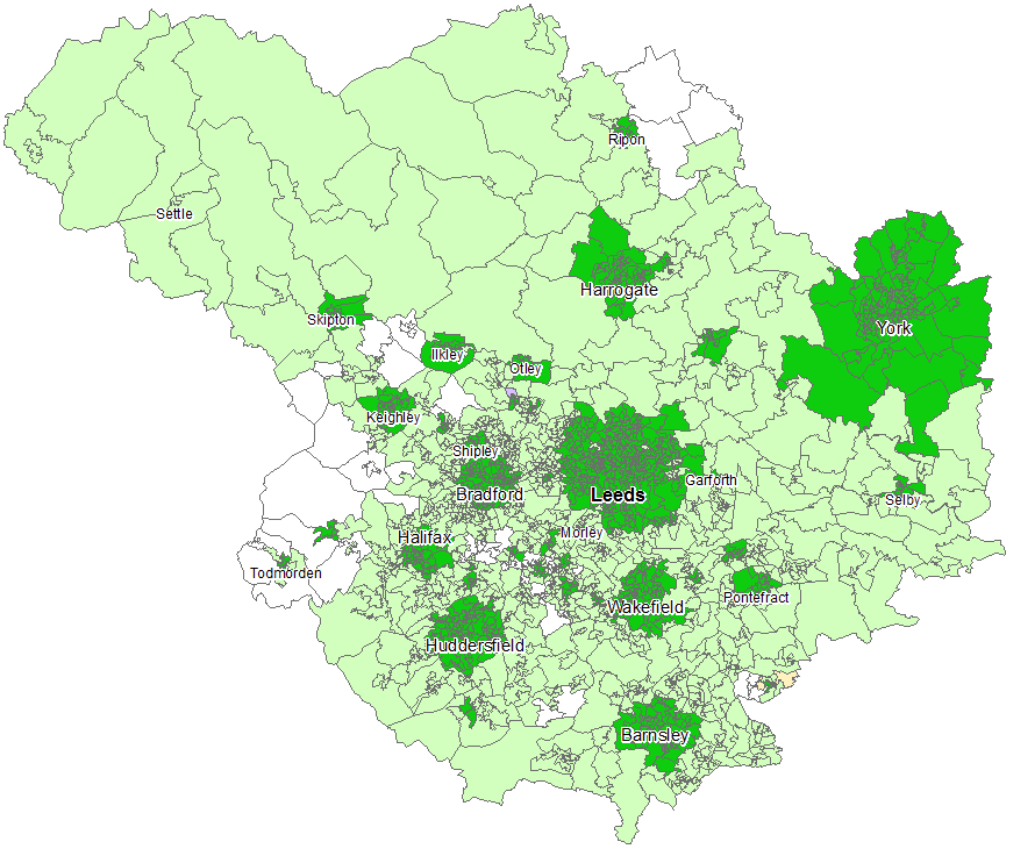
```
dir.create("results") # Create a new folder named results
writeOGR(origins, "results", "huff_probs1", driver = "ESRI Shapefile") # Save the origins shapefile in
```

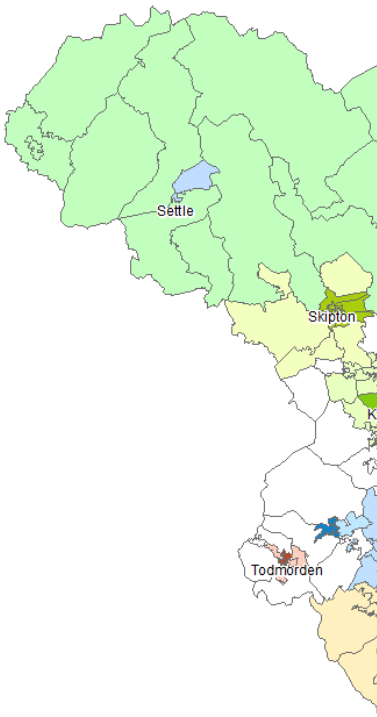
Or we could extract each retail centre catchment as individual polygon and save it in a folder using the 'get_catchment' function from huff-tools. The get_catchment function accepts 9 arguments and outputs a shapefile. The arguments are:
a) The previously created shapefile (poly).
b) The path to save the output of the function (path_to_save).
c) A name for the output shapefile (shp_name_out).
d) The name of the destination (town_centre).
e) The upper bound of the Huff probability (huff_prob_upper).
f) The lower bound of the Huff probability (huff_prob_lower).
g) The name of the "origins" field in the shapefile (origins_name).
h) The name of the "destinations" field in the shapefile (destinations_name).
i) The name of the huff probabilities field in the shapefile (huff_probability).
The last 3 arguments are optional as the default field names produced by the 'huff_basic' method are used.

```
towncentre <- as.character(origins@data$destinations_name[1])
towncentre

# Extract primary catchment (i.e. Huff probability >= 0.5)
get_catchment(poly=origins, path_to_save="results", shp_name_out=towncentre, town_centre=towncentre, hu
```

The saved shapefile contains the patronage probabilities for each retail centre in Leeds City Region. All we need to do now is to open it with any GIS package such as QGIS or ArcGIS and visualise the estimated Huff patronage. If we use values > 0.5 for primary catchments and values > 0.2 for secondary catchments, then we should have the following output, referred to as the basic model.


Ripon
Settle
Harrogate
York
Skipton
Ilkley
Otley
Keighley
Shipley
Garforth
Bradford
Leeds
Selby
Halifax
Morley
Todmorden
Pontefract
Wakefield
Huddersfield
Barnsley



Using distinctive colours for each town centre we end up with even a more clear picture

4. Estimating the characteristics of population flows to retail centres.

4.1 Huff probabilities and relationship with area.

In the next part of the presentation we will look at a few ways of analysing the estimated Huff probabilities and exploring their relationship with area and population. It might also be interesting to aggregate the results of our analysis at the retail centre hierarchy level. For this, we need to merge the attractiveness score data with the origins locations data.

```
# Add Rank field to origins
origins@data <- data.frame(origins@data,
                           attr_score[match(origins@data$destinations_name,
                                            attr_score$ID), c("ID", "Rank")])
origins@data$ID <- NULL
```

A common task in spatial analysis, is summarising and manipulating spatial information. For example we

might want to calculate the total area of the Leeds City Region, which requires accessing each polygon of the 'origins' dataset and extracting the 'area' slot. An efficient way of repeatedly applying a task in R is by using one of the functions of the 'apply' family. In the next snippet you will see how to use the 'sapply' function, which requires two arguments, first a dataset with multiple elements, and then a function to apply to each element.

```
total_area <- sum(sapply(origins@polygons, function(x) x@area)) / 1000^2
```

Using the above R code we loop over each polygon, extract its area which is then stored in a vector (that is the output of the 'sapply' function) and we then use the 'sum' function to obtain the sum of the values in that vector. We can then calculate the entire/primary catchment areas for each level of the retail centre hierarchy in a similar manner that we calculated the total area . In this case, however, we need to slice the data based on the 'Rank' and 'huff_probability' fields before we can extract the area of each polygon. So we need to add one more 'sapply' function to loop over the values of the 'Rank' field and we should also specify the Huff probability values that define the entire/primary catchment areas.

```
# Calculate entire catchment area for each level of town centre hierarchy
sapply(1:4, function(x) sum(sapply(origins[origins@data$Rank == x &
                                           origins@data$huff_probability >= 0.2, ]@
                                     polygons, function(y) y@area))/1000^2)
```

```
## [1] 4541.02337  588.14575  255.40046   79.00026
```

```
# Calculate primary catchment area for each level of town centre hierarchy
sapply(1:4, function(x) sum(sapply(origins[origins@data$Rank == x &
                                           origins@data$huff_probability >= 0.5, ]@
                                     polygons, function(y) y@area))/1000^2)
```

```
## [1] 686.15163 143.87776  87.99003  35.12902
```

```
# Also look at the number of LSOAs per town centre hierarchy level
table(as.factor(origins@data$Rank))
```

```
##
##    1    2    3    4
## 1092  430  198  121
```

```
# Get number of retail centres per hierarchy level
table(as.factor(destinations@data$Rank))
```

```
##
##  1  2  3  4
##  5  7 16 21
```

As can be seen from the above results, the catchment areas are greater for the retail centres higher up in the hierarchy even though they account for less than a quarter of all the retail centres in the study area. Of course this should be expected, nevertheless, it is useful to have this information particularly if we want to compare the output of different Huff models (e.g. when we test the effect of changing the beta and alpha values) as it will be shown in the next part of the presentation.

4.2. Huff probabilities and total population patronising a retail centre.

Similar to the previous example we can look at the relationship between population and the estimated Huff probabilities. For this we will use population data from the 2011 Census. We need to import first the population data and then merge these with the Huff probabilities data. By multiplying the population by the Huff probabilities we can estimate the potential number of people in each LSOA patronising each retail centre. Finally, by using the 'aggregate' function we can calculate the total population patronising each retail centre.

```
# Import population data
pop <- read.csv("~/Dropbox/liverpool/retail/PresentationLDC/data/LCR_pop.csv")
names(pop)[1] <- "origins_name"

# Join Huff probabilities and population data
huff_probs <- inner_join(huff_probs, pop)
```

```
## Joining, by = "origins_name"
```

```
# Estimate the ratio of population in a LSOA patronising a given retail centre
huff_probs$pop_by_huff <- huff_probs$huff_probability * huff_probs$all_pop

# Check that the sum of population ratios is equal to total population in study area
sum(huff_probs$pop_by_huff)
```

```
## [1] 2952057
```

```
sum(pop$all_pop)
```

```
## [1] 2952057
```

```
# Total potential customer flow to each destination
destination_pop_flows <- aggregate(huff_probs$pop_by_huff,
                                   by = list(huff_probs$destinations_name), sum)
names(destination_pop_flows) <- c("destinations_name", "total_pop")

str(destination_pop_flows)
```

```
## 'data.frame':    49 obs. of  2 variables:
##  $ destinations_name: Factor w/ 49 levels "TC0010","TC0033",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ total_pop        : num  10775 12159 121978 20060 13651 ...
```

The population flows for each retail centre are shown below

5. Evaluating the effect of different parameter values

In this part of the presentation we willl look at the effect that using different parameter values might have to the output of the Huff model.

To start with, we increase all beta parameters by a constant value of 0.1, so we have beta=1.5 for rank 1, beta=1.7 for rank2, beta=1.9 for rank 3 and beta=2.1 for rank 4. The output shown below implies that when

Console | R Markdown

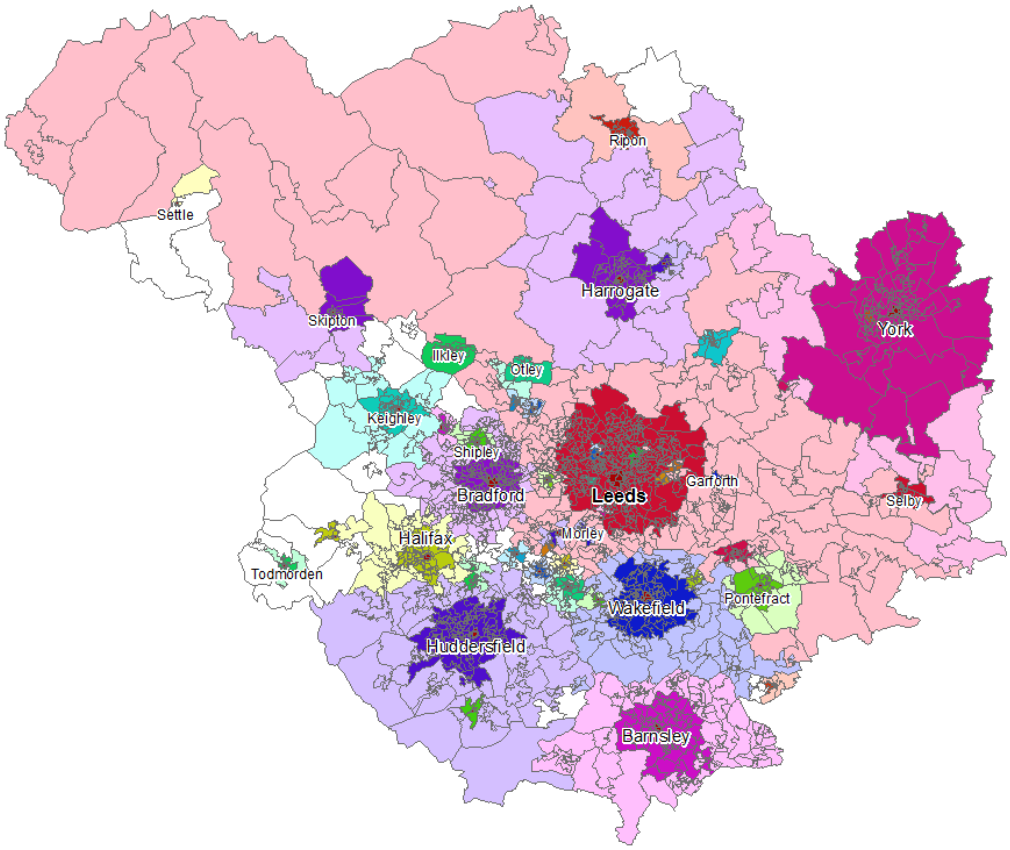
~/Liverpool/Stat/LeedsCR/PresentationLDC/

	ID	Name	total_pop	Centre_Type	Rank
28	TC0731	Leeds	650738.370	Town centre	1
8	TC0167	Bradford	312408.834	Town centre	1
24	TC0662	Huddersfield	263090.588	Town centre	1
49	TC1434	York	254612.313	Town centre	1
44	TC1292	Wakefield	253706.219	Town centre	1
3	TC0076	Barnsley	163295.909	Town centre	1
19	TC0571	Harrogate	149424.884	Town centre	1
16	TC0552	Halifax	111020.108	Town centre	2
26	TC0681	Keighley	76993.527	Town centre	2
46	TC1370	White Rose Centre	53852.659	Shopping Centre	2
34	TC0968	Pontefract	50289.265	Town centre	2
13	TC0390	Dewsbury	44562.908	Town centre	3
33	TC0930	Otley	38550.810	Town centre	2
10	TC0255	Castleford	34040.595	Town centre	3
39	TC1093	Shipley	31609.239	Town centre	3
40	TC1108	Skipton	30591.780	Town centre	2
9	TC0183	Brighouse	29997.261	Town centre	3
20	TC0592	Headingley	29857.955	Town centre	3
29	TC0851	Morley	25377.121	Town centre	3
22	TC0601	Heckmondwike	24085.728	Town centre	3
37	TC1074	Selby	22626.943	Town centre	3
7	TC0127	Birstall Shopping Park	22297.620	Shopping Centre	3
4	TC0086	Batley	21761.171	Town centre	3
25	TC0674	Ilkley	20756.291	Town centre	3
35	TC0995	Pudsey	19232.308	Town centre	3
11	TC0310	Cleckheaton	18700.466	Town centre	3
36	TC1027	Ripon	17568.415	Town centre	3
45	TC1359	Wetherby	15164.490	Town centre	3
27	TC0718	Knaresborough	14236.359	Town centre	3
5	TC0121	Bingley	13101.824	Town centre	4
18	TC0563	Harehills	12596.310	Town centre	4
12	TC0364	Crossgates	12453.060	Town centre	4
21	TC0600	Hebden Bridge	11312.716	Town centre	3
32	TC0928	Ossett	11018.868	Town centre	4
48	TC1431	Yeadon	10033.192	Town centre	4
1	TC0010	Acomb	9703.752	Town centre	4
2	TC0033	Armley	9056.538	Town centre	4
30	TC0893	Normanton	8231.867	Town centre	4
23	TC0641	Holmfirth	7896.598	Town centre	4
15	TC0543	Guiseley	7429.403	Town centre	4
41	TC1115	South Elmsall	6479.231	Town centre	4
43	TC1249	Todmorden	5879.575	Town centre	4
47	TC1409	Wombwell	5495.162	Town centre	4
17	TC0554	Halton, Leeds	5115.026	Town centre	4
14	TC0499	Garforth	5109.641	Town centre	4
6	TC0126	Birstal Smithies	3876.055	Town centre	4
31	TC0918	Oakwood, Leeds	3331.757	Town centre	4
42	TC1218	Tadcaster	2311.388	Town centre	4
38	TC1076	Settle	1174.902	Town centre	4

>

Figure 4: Pop flows by retail catchments

comparing this model to the basic model some differences are visible, although it could be argued that they are



negligible.

For the next model, we decrease the gap between the beta values for each rank from 0.2 in the above model to 0.1. So we use the following beta values: 1.4 for rank 1, 1.5 for rank 2, 1.6 for rank 3 and 1.7 for rank 1.8. By doing so, we increase the complex (non-linear) effect of the competition between different types of retail centres. The output, shown below shows clearly that the importance of Leeds as the major centre in the region has been enhanced significantly. Simultaneously, the role of smaller centres (rank 3 and 4) has been minimised , which had an effect on the extent of their catchments.

The third example uses Huff's default values of beta=2 and alpha=1 across all retail centres, omitting the disaggregation by centre type. Effectively, the hierarchy within retail centres is accounted solely by the Composite Attractiveness Index. As the output shows, such calibration has a considerable effect on the catchments of both smaller centres and the biggest one - Leeds, increasing the extent in the case of the former but decreasing of the latter. It is mainly due to the fact that the distance decay for all types of centres is identical. Although it seems unrealistic, it may be difficult to decide with a large degree of confidence what is the best scenario as such model should be validated against real world data.

It is obvious, thus, that a model calibration can have a significant impact not only on the extent of retail catchments but also on the predicted patronage levels. The table below puts that into context by using the

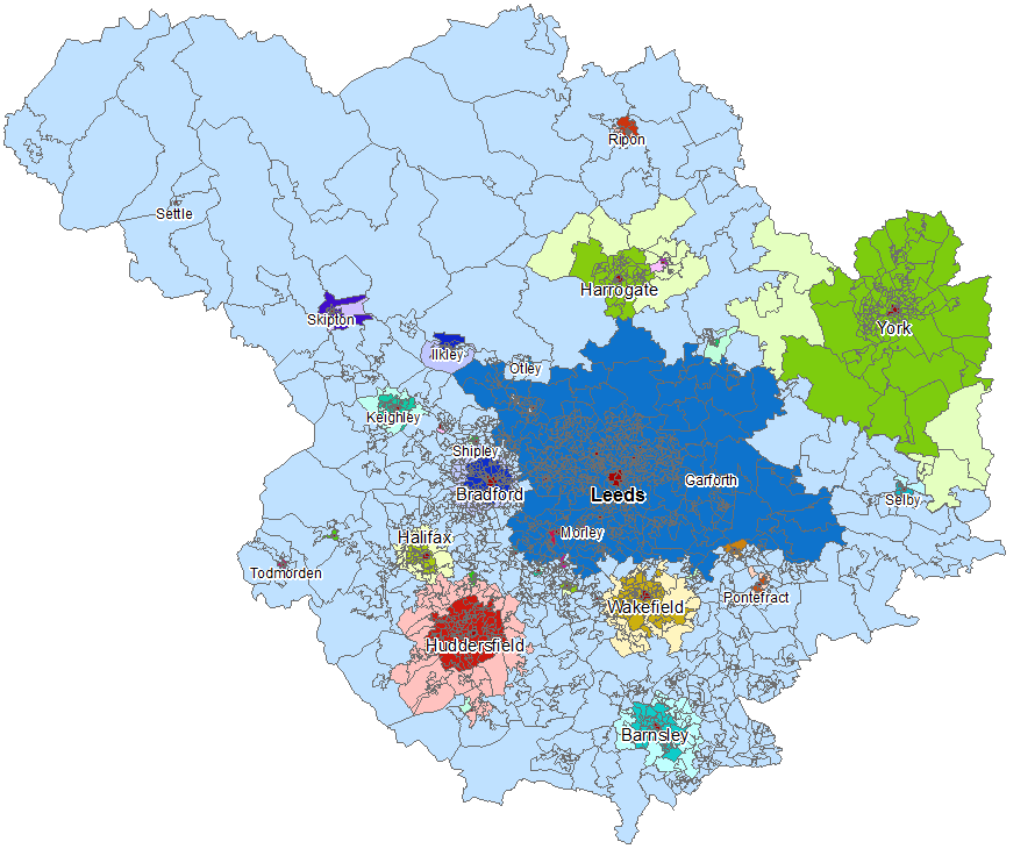


Figure 5: Retail catchments with altered beta values - example 2

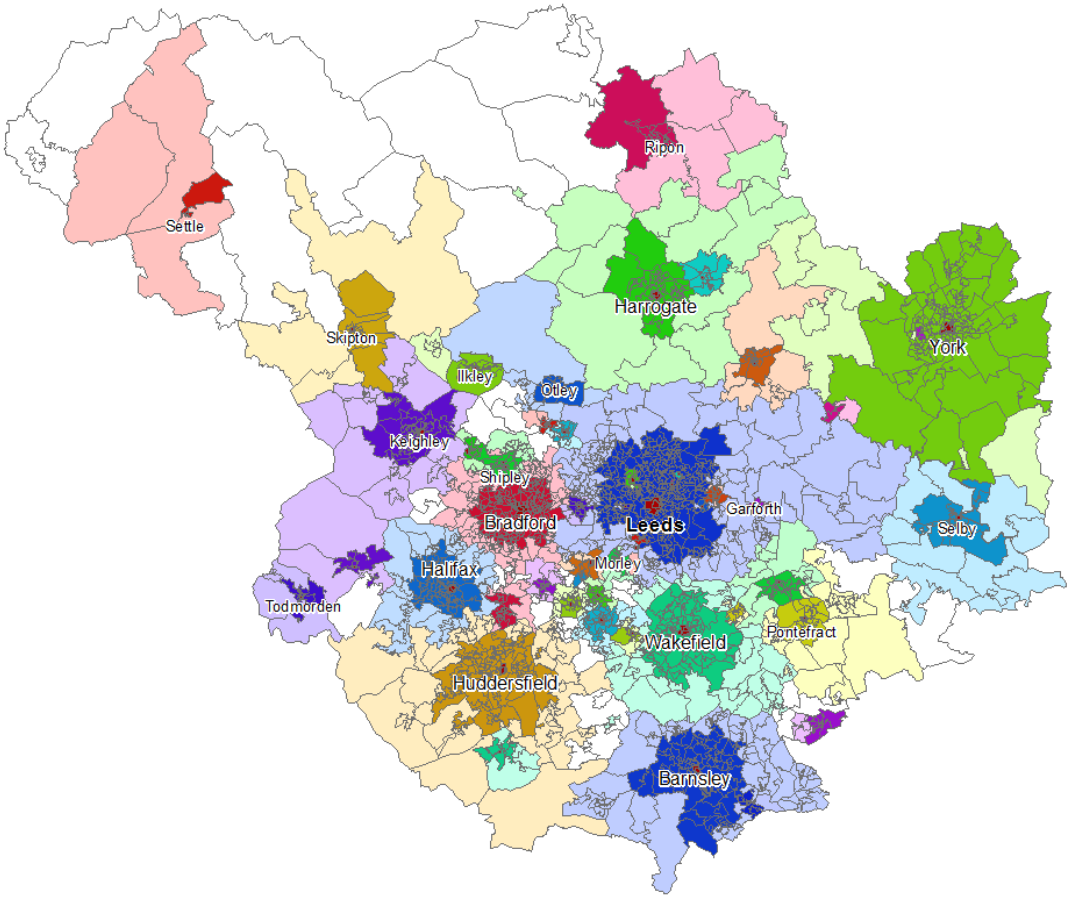


Figure 6: Retail catchments with altered beta values - example 3

estimated area and population patronising a retail centre, similar to what was shown in section 4 of this presentation. More specifically, 5 different models are presented, aggregated at the retail hierarchy level. For simplicity, the population flows have been calculated by allocating each LSOA population to the retail centre with the highest probability of being patronised from that particular LSOA.

		Area		Population	
Total		5723.483 Km^2		2952.057 (000)	
Model	Rank	Primary (%)	Entire (%)	Primary (%)	Entire (%)
Base Model beta = 1.4, 1.6, 1.8, 2.0 alpha = 1, 2	1	686.151 (11.99)	4541.023 (79.34)	685.113 (23.21)	995.147 (33.71)
	2	143.877 (2.51)	588.146 (10.28)	235.147 (7.96)	374.820 (12.70)
	3	87.990 (1.54)	255.400 (4.46)	131.818 (4.46)	195.230 (6.61)
	4	35.129 (0.61)	79.000 (1.38)	86.020 (2.91)	121.892 (4.13)
Model 2 beta = 1.3, 1.5, 1.7, 1.9	1	636.871 (11.12)	4719.075 (82.45)	640.982 (21.71)	992.220 (33.61)
	2	105.629 (1.84)	469.045 (8.19)	198.689 (6.73)	336.863 (11.41)
	3	78.210 (1.37)	207.872 (3.63)	118.677 (4.02)	174.481 (5.91)
	4	25.593 (0.45)	74.796 (1.31)	72.595 (2.46)	108.520 (3.67)
Model 3 beta = 1.5, 1.7, 1.9, 2.1	1	731.523 (12.78)	4313.029 (75.36)	720.313 (24.40)	1002.991 (33.98)
	2	162.592 (2.84)	616.153 (10.77)	268.618 (9.10)	407.091 (13.79)
	3	119.671 (2.09)	333.262 (5.82)	156.481 (5.30)	217.914 (7.38)
	4	40.806 (0.71)	113.514 (1.98)	96.590 (3.27)	137.499 (4.66)
Model 4 alpha = 1	1	687.551 (12.01)	4543.134 (79.38)	690.036 (23.37)	1002.572 (33.97)
	2	143.878 (2.51)	588.146 (10.28)	233.856 (7.92)	373.529 (12.65)
	3	87.990 (1.54)	255.400 (4.46)	125.795 (4.26)	189.208 (6.41)
	4	32.876 (0.57)	76.889 (1.34)	67.289 (2.28)	107.599 (3.64)
Model 5 Distance to Centroids	1	625.363 (10.93)	4577.796 (79.98)	590.743 (20.01)	944.805 (32.01)
	2	93.839 (1.64)	523.562 (9.14)	159.613 (5.41)	319.587 (10.83)
	3	70.773 (1.24)	220.015 (3.84)	90.258 (3.06)	152.259 (5.16)
	4	21.607 (0.38)	71.220 (1.24)	47.843 (1.62)	89.112 (3.02)

Figure 7: Table comparing catchments area and population for different scenarios