Cluster Analysis in R

Energy efficiency in the private rental sector in Greater London

Like many countries globally, the private rental sector in England contains some of the poorest quality and least energy efficient properties. In this exercise we will use cluster analysis (specifically a k-means clustering approach) to identify patterns in energy inefficiency in the private rental sector across Greater London. We will use new energy performance certificate (EPC) data that offers detailed information about the energy efficiency of properties.

The exercise is especially topical as the government in England is currently consulting on a new minimum energy efficiency standards for the private rental sector. Currently, private rentals must have a rating of E, however, there are ongoing consultations about increasing the rating to C from 2025.

K-means cluster analysis

Today's exercise builds on a long tradition of geodemographic research in geography. Here, k-means clustering is applied to identify clusters of data points with similar attributes. The technique is widely applied to understand all sorts of phenomena - including COVID-19, digital infrastructure, cities, ecology, and poverty.

This exercise is an adaptation of a wider piece of analysis that develops an energy inefficiency classification for the private rental sector across the whole of England and Wales. The initial results of that analysis are available here, to give you a sense of the type of outputs you are aiming for!

You will also find that some of the skills in the last R Ladies Edinburgh session about Spatial Data with Alessia Calafiore might come in useful!

Install libraries

We install the packages and libraries for our analysis. The descriptions provide an overview of what each group of packages will help you to achieve.

```
library(tidyverse) # Organising data
library(dplyr)
library(corrplot) # Plotting data
library(ggplot2)
library(sf) # Working with spatial data
library(tmap)
library(cluster) # Cluster analysis
library(factoextra)
library(RColorBrewer) # Colour palettes
```

Download the data

First things first, **set the working directory** using *setwd()*. Here we will store the underlying data and outputs. Into the working directory, **download the dataset** *greaterlondon_efficiency.gpkg*, available available from dropbox.

The file is a **geopackage for working with spatial data**. The geopackage provides different energy efficiency characteristics for Output Area in Greater London. Each Output Area represents approximately 125 households. These Output Area boundaries are based on the recent 2021 Census.

Because the data is spatial, use the st_read function to read it. You can see details about the dataset in the messages. The dataset for Greater London include 26369 Output Areas.

```
gl_efficiency <- st_read("greaterlondon_efficiency.gpkg")

## Reading layer `greaterlondon_efficiency' from data source
##   `C:\Users\jj22511\OneDrive - University of Bristol\Rladies\greaterlondon_efficiency.gpkg'
##   using driver `GPKG'
## Simple feature collection with 26369 features and 11 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 503568.1 ymin: 155850.8 xmax: 561957.4 ymax: 200933.9
## Projected CRS: OSGB36 / British National Grid
```

List the variables

Using the *colnames* function, investigate the list of variables. Here you will see a list of eleven variables, plus an additional *geom* variable that provides spatial information.

```
colnames(gl_efficiency)
```

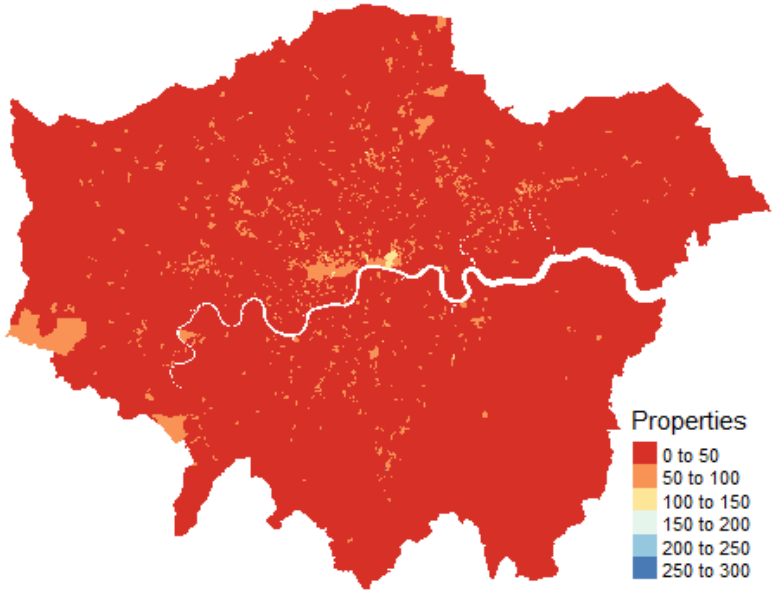
The variables provide information about properties in each Output Area, drawn from Energy Performance Certificates. You can find out more about the dataset here. EPC data gives a rating for energy efficiency, from A to G, with A being the highest of efficiency. The certificates also provide information about the property age, type, and energy efficiency characteristics. The table describes each of our variables.

Variable code	Variable name	Type
privaterentals	Private rental properties	Count
built_pre1900	Properties built before 1900	Count
built_since2012	Properties built after 2012	Count
rating_current_Dandbelow	Properties with a current EPC rating of D or below	Count
rating_potential_Dandbelow	Properties with a potential EPC rating of D or below	Count
property_allterrace	Terraced properties	Count
property_housebungalow	Houses and bungalows	Count
property_flatmaisonette	Flats and maisonettes	Count
efficiency_hotwater	Inefficient hot water system	Count
efficiency_walls	Inefficient walls	Count
efficiency_mainsheat	Inefficient mains heating system	Count

Map the variables

Next, we map the data to visualise its extent across Greater London. We use the *tm_shape* function, part of the *tmap* package, to map the count of private rental properties for Output Areas. But you can try out different variables of interest by replacing *privaterentals in tm_fill*.

```
tm_shape(gl_efficiency)+
 tm_fill(col = "rating_current_Dandbelow",
       title = "Properties",
       palette = "RdYlBu")+
 tm_layout(frame = FALSE)
```



Standardise the variables

Once we are happy that the data has downloaded correctly, we standardise each variable. Standardisation ensures that the variables are comparable, even if they are provided in different units, or have a skewed distribution. Even though our variables are all provided as counts, some of the variables have much higher values than others, making standardisation necessary.

We use *scale*, a base R function that **converts values into z-scores**. A z-score tells us how many standard deviations away a given value lies from the mean. Whilst in a spatial *sf* format, our data can't be standardised, so we first use *st_drop_geometry* to convert the data into a standard data frame.

```
gl_efficiency_df <- st_drop_geometry(gl_efficiency)
gl_efficiency_df <- scale(gl_efficiency_df)
```

Explore relationships between variables

The *corrplot* package generates a correlation matrix that tells us about the relationship between the variables. It allows us to assess whether it is useful to retain all the variables, or to highlight if some variables are telling the same story and should therefore be excluded from the analysis.

First, we **rename our variables**, and calculate the **correlations between variables** using *pearson* correlation. The Pearson correlation method is the most common method to use for numerical variables. It assigns a value between − 1 and 1, where 0 is no correlation, 1 is total positive correlation, and − 1 is total negative correlation.

```
# Rename columns for the plot
colnames(gl_efficiency_df) <- c("Private rental properties",
                                "Built pre-1900",
                                "Built since 2021",
                                "Current rating D and below",
                                "Potential rating D and below",
                                "Terrace",
                                "House or bungalow",
                                "Flat or maisonette",
                                "Inefficient hot water",
                                 "Inefficient walls",
                                "Inefficient mains heating")

# Calculate correlation
corr = cor(gl_efficiency_df, method = "pearson")
```

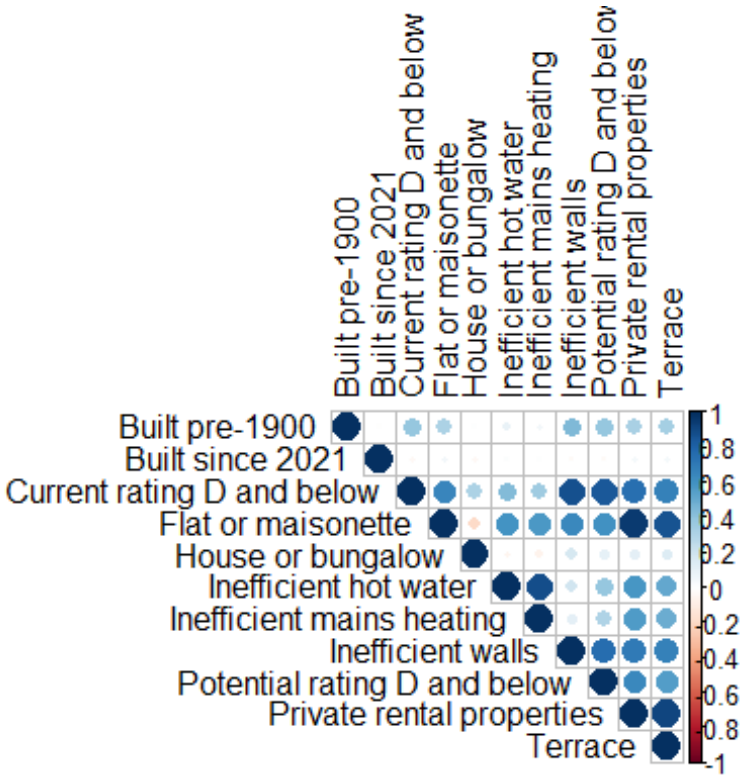
We then **plot our correlation matrix**. There are lots of changes you can make to the correlation plot that you might like to experiment with. You can find out more here but common changes include:

- setting the method: try *square, ellipse, number, shade, pie,* or *color*
- setting the type: try *full* or *lower*
- setting the order: try *AOE, FPC,* or *hclust*

The correlation plot measures the strength (the size of the circle) and type (positive or negative) of the relationship. We might then choose to remove some variables before analysis if any variables are highly correlated.

Are any variables highly correlated, that you might consider removing?

```
# Plot correlation matrix
corrplot(corr = corr,
      method = "circle",
      type = "upper",
      order = "alphabet",
      tl.col = "black", # Labels set as black
      sig.level = 0.05)
```



Decide on the number of clusters

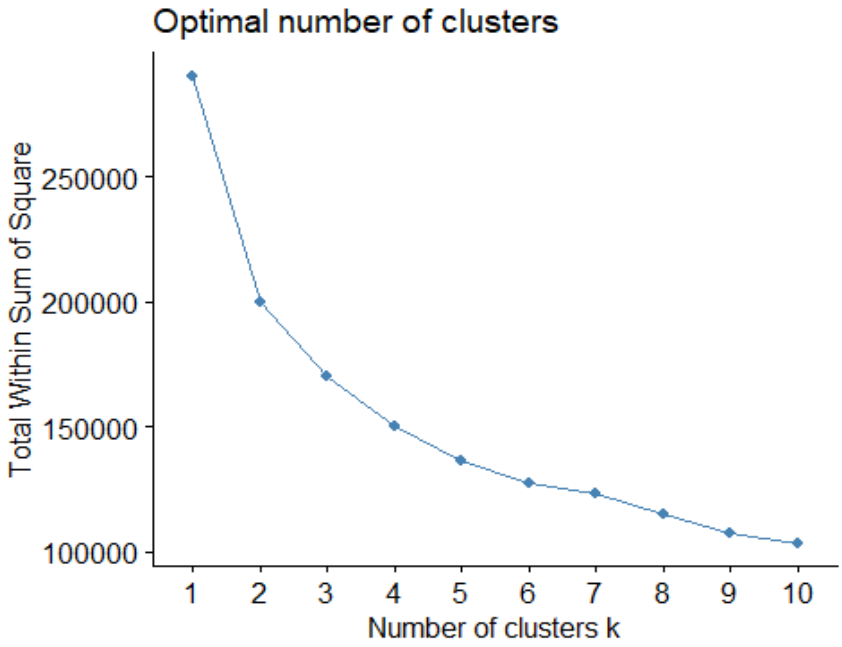
Deciding on an appropriate number of clusters (*k*) is perhaps the most important decision made in the analysis. There is no right number of clusters. We decide using different diagnostics coupled with our own knowledge about the variables analysed.

The *factoextra* package includes a range of useful functions. The package is relatively computationally intensive, so we will only calculate one in this exercise - the snappily named **Within-Cluster-Sum of Squared Errors (WSS).** However, there are other diagnostics you might explore.

The WSS calculates the squared error for each data point. This is the square of the distance between the Output Area and the predicted cluster center. The WSS score is the sum of the squared errors for all the Output Area in the dataset. We calculate the WSS for solutions with different numbers of clusters (*k*), selecting the cluster where the WSS first starts to diminish. This is signified by a kink, or elbow, in the plot, indicating the point at which adding more clusters is less likely to be beneficial.

To compute the WSS, we set an initial seed at which the analysis will begin. This ensures our analysis will be the same each time. We use the *fviz_nbclust()* function, and opt for a *wss* method.

```
set.seed(125) # Set initial seed
fviz_nbclust(gl_efficiency_df, kmeans, method = "wss") # Compute total within-cluster sum of square
```



Based on the WSS there is no clear "elbow" in the plot. We will therefore experiment with 5, 6 and 7 cluster solutions to assess which is best.

If you have time, you could change the method to *bss* to experiment using a Between-Cluster-Sum-of-Squares approach. You might also carry out a Silhouette analysis to estimate the average distance between clusters using *fviz_silhouette()*. You could also change the seed, to see how this effects the results.

Generate k-means solutions

We are now ready to generate some clusters!

K-means clustering is **a partitioning-based technique**, that iteratively relocates data points between a set of k clusters. Each observation belongs to the cluster with the nearest mean. It is helpful to explore the structure of large multivariate datasets, reducing their complexity. The goal of k-means clustering is to produce groups of datapoints that have a high degree of similarity within a group, and a low degree of similarity between groups. It provides both qualitative and quantitative insight into the data, rather than providing a definite set of groupings for the data points.

For this exercise we use the *kmeans* function from the *stats* package. The process is as follows:

- We first specify the number of clusters (k) using *centers*
- Each observation is assigned to their closest centre, based on the distance between the object and centre
- Cluster centres are updated based on the new mean of values for all the data points in the cluster
- This process is iterated through until the points stop changing their cluster assignment. Here we set the maximum number of iterations to 1000, but you could opt for a lower number of iterations if you like.

```
# Generate clusters for 5, 6 and 7 clusters
kmeans_5 <- kmeans(gl_efficiency_df,
                   centers = 5,
                   iter.max = 1000)

kmeans_6 <- kmeans(gl_efficiency_df,
                   centers = 6,
                   iter.max = 1000)

kmeans_7 <- kmeans(gl_efficiency_df,
                   centers = 7,
                   iter.max = 1000)
```

The **size of clusters** is also a useful diagnostic. Researchers often recommend that clusters should be a similar size. However, in our case, private rental properties tend to concentrate spatially in particular areas (as we know from our first map above!) so some smaller clusters may be appropriate.

How do the three different cluster solutions vary in size?

```
kmeans_5$size
kmeans_6$size
kmeans_7$size
```

Visualise cluster centres

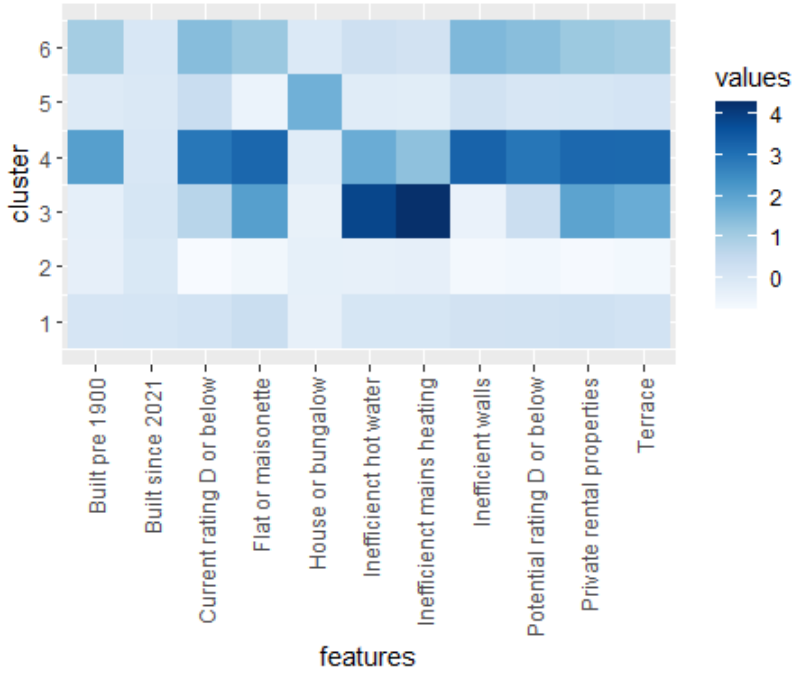
To understand more about the unique characteristics of each of our clusters, it is useful to examine the mean for different underlying variables. We can **generate a heat map** using functions from the *ggplot* package. Remember, the values reflect the standardised values.

From the heat map we can see that clusters three and four tend to have the highest values across the majority of the variables included.

What other characteristics can you identify that help to differentiate the six clusters from one another?

```
# Define the center of each cluster
cluster <- c(1: 6)
center <- kmeans_6$centers
center_df <- data.frame(cluster, center)

# Reshape the data into a long format
center_reshape <- gather(center_df, features, values, Private.rental.properties:Inefficient.mains.heating)

# Edit the labels
center_reshape <- center_reshape %>%
  mutate(features = recode(features,
                           "Private.rental.properties" = "Private rental properties",
                           "Built.pre.1900" = "Built pre 1900",
                           "Built.since.2021" = "Built since 2021",
                           "Current.rating.D.and.below" = "Current rating D or below",
                           "Potential.rating.D.and.below" = "Potential rating D or below",
```

```
                "Terrace" = "Terrace",
                "House.or.bungalow" = "House or bungalow",
                "Flat.or.maisonette" = "Flat or maisonette",
                "Inefficient.hot.water" = "Inefficienct hot water",
                "Inefficient.walls" = "Inefficient walls",
                "Inefficient.mains.heating" = "Inefficienct mains heating"))
# Specify the colour palette
hm.palette <-colorRampPalette(brewer.pal(9, 'Blues'), space='Lab')

# Plot the centers for each variable
ggplot(data = center_reshape, aes(x = features, y = cluster, fill = values)) +
  scale_y_continuous(breaks = seq(1, 14, by = 1)) +
  geom_tile() +
  coord_equal() +
  scale_fill_gradientn(colours = hm.palette(90)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



Map final clusters

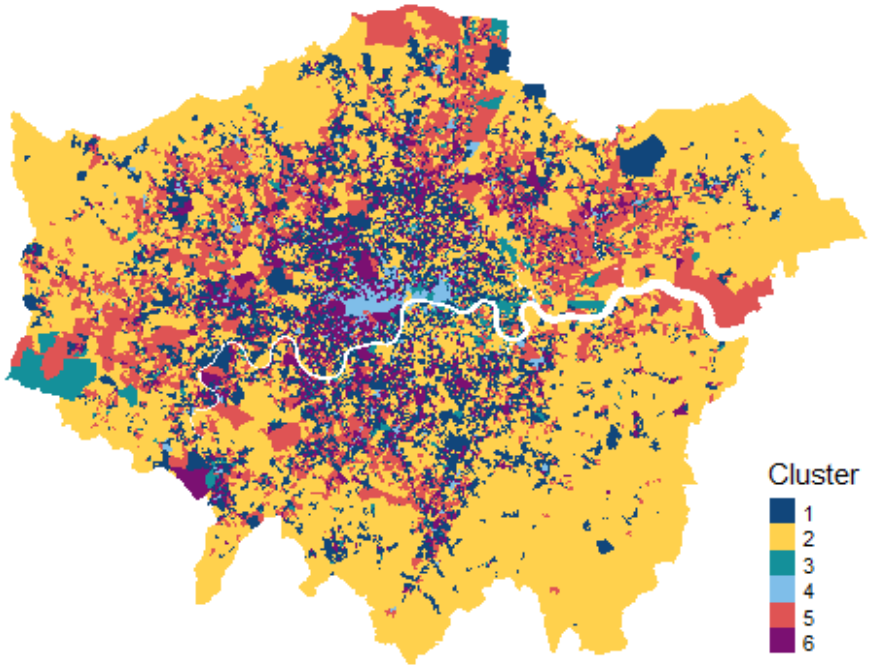
To finish, we add the cluster numbers for each Output Area to the original dataset.

```
# Add cluster information to original data
gl_efficiency_clusters <- gl_efficiency

gl_efficiency_clusters$cluster_5 <- as.character(kmeans_5$cluster)
gl_efficiency_clusters$cluster_6 <- as.character(kmeans_6$cluster)
gl_efficiency_clusters$cluster_7 <- as.character(kmeans_7$cluster)

# Convert back to sf format
gl_efficiency_clusters_sf <- st_as_sf(gl_efficiency_clusters)
```

This allows us to **map the data**. Here we use the *tm_shape* and *tm_fill* functions from the *tmap* package. We map the six cluster solution, using a range of contrasting colours. You could experiment with mapping a different number of clusters, to see how the solutions vary spatially.

How does the distribution of clusters change if you choose a solution with a large number of clusters?

```
tm_shape(gl_efficiency_clusters_sf)+
 tm_fill("cluster_6",
         title = "Cluster",
         palette = c( "1" = "#11467b",
     "2" = "#ffd14d",
     "3" = "#14909a",
     "4" = "#7fbee9",
     "5" = "#df5454",
     "6" = "#7b1072"))+
 tm_layout(frame = FALSE)
```



Finally, you might like to decide on some **names for your clusters**. These names might reflect the underlying variables that are highest or lowest in each cluster (e.g., inefficient, older, flats), or the spatial distribution of the clusters (e.g., central, concentrated, peripheral, rural).

What would you name the clusters in your map, based on the underlying data or spatial distribution?

Funding: The exercise is based on research part of *Mapping Ambient Vulnerabilities* UKRI Future Leaders Fellowship (MR/V021672/1).

To find out more please get in touch with Caitlin Robinson.

Mapping Ambient Vulnerabilities