GY638 - Housing Data Project

Sean O'Riogain (18145426)

17 May 2019

```
knitr::opts_chunk$set(echo = TRUE)
getwd()
```

```
## [1] "C:/Users/oriogain/Dropbox/Maynooth/NCG613 - Data Analytics Project/Project - Solo"
```

```
# install.packages("sqldf")                         # Install needed packages
# install.packages("mlbench")                       # (One time only)
# install.packages("doParallel")
# install.packages("randomForest")
# install.packages("caret")

suppressPackageStartupMessages(library(tidyverse))  # For deplyr functions etc.
suppressPackageStartupMessages(library(sqldf))      # For sqldf() etc.

suppressPackageStartupMessages(library(sp))         # For SpatialDataFrame() etc.
suppressPackageStartupMessages(library(rgdal))      # For readOGR() etc.
suppressPackageStartupMessages(library(rgeos))      # For mapping etc.
suppressPackageStartupMessages(library(classInt))   # For ClassIntervals()
suppressPackageStartupMessages(library(RColorBrewer)) # For Palette() etc.
suppressPackageStartupMessages(library(GISTools))   # For auto.shading() etc.
suppressPackageStartupMessages(library(wrapr))      # For match_order() etc.

suppressPackageStartupMessages(library(gclus))      # For cpairs() etc.

suppressPackageStartupMessages(library(mlbench))    # For Random Forest functions etc.
suppressPackageStartupMessages(library(randomForest)) # For Random Forest functions etc
suppressPackageStartupMessages(library(caret))      # For RFE functions etc.

suppressPackageStartupMessages(library(parallel))   # For parallelising RFE
suppressPackageStartupMessages(library(doParallel)) # For parallelising RFE
```

London House Price Estimation Application (Proof of Concept)

Executive Summary

Objectives

Outlier Analytics Ltd. (OAL) has been engaged by our client, Charlton Estates Ltd. (CEL), to conduct a proof of concept (PoC) exercise whose aim is to demonstrate the feasibility of eventually building an application capable of accurately estimating the selling/purchase price of a residential property in any of the London Boroughs.

Background

CEL is a large estate agency that specialises in the valuation and sale of residential property exclusively within the London Boroughs. It operates a branch network of 32 offices across this territory and employs a total workforce of 295 people. With the advent of Brexit, CEL has been losing some of its most experienced agents, who are almost exclusively EU nationals returning to their home countries, and believes that this trend is only going to increase as the exit date approaches. It is also aware that many of its competitors are experiencing the same difficulties. Therefore, CEL predicts that replacing them is going to be extremely difficult (if not impossible) to do at an affordable cost in London's post-Brexit labour marketplace.

This has led CEL to the conclusion that it will need to use technology to enable its less experienced local staff - both current and future - to effectively perform the type of type of work that its lost staff used to do. In particular, it needs to provide them with a tool that can accurately estimate the value (i.e. selling/purchase price) of any residential property within its target market given its key attributes - its location and floor area, in particular.

To this end, CEL has provided OAL with a Microsoft Excel spreadsheet that contains the details of 12,536 residential properties that it has sold during the past 12 months. This data includes the selling/purchase price, location and floor area information as well as other details which CEL believes could be helpful during the valuation process. Because this data is commercially sensitive, the provision of this data is subject to a CEL non-disclosure agreement which has been duly signed by OAL.

Business Requirements

As CEL has yet to be convinced as to the ability of modern Machine Learning techniques to deliver the type of capability referred to above,it is only prepared to fund the development of a proof-of-concept prototype (PoC) at this stage for which it has provided OAL with the following requirements (both functional and non-functional):

1. Given the location and size of a residential property in any of the London Boroughs, the PoC should be able to estimate its selling/purchase price (expressed as a price range with a Â£50,000 interval) with a 70%+ accuracy rate.

   Also note that, although the following are not requirements for this PoC exercise, they will be **for any future production application**:

   a. The price range interval must be configurable.

   b. The sizes of properties in the dataset provided by CEL are in square meters but it must be configurable to use square feet instead depending on its user's preference.

   c. The approximate location of the properties in that dataset are in Easting and Northing coordinate format but it must be capable of using GPS coordinates.

   d. It must also be configurable to make use of additional house price data as they become available.

2. The PoC should provide some level of confidence that such an estimate could be produced with sub-second response time.

3. The PoC code must be capable of running on a Windows 10 workstation or laptop.

   - Note that any eventual application must also be capable of running on other platforms - especially on mobile devices, such as smart phones and tablets on the then current Microsoft, Apple and Google operating system versions.

4. As CEL outsources the vast bulk of its IT services, it is open to the use of any coding language for developing the PoC.

   - However, it does wish to be consulted during the choice of such technologies for any resultant production application.

5. While CEL is open to the use of any Machine Learning technique which achieves the required level of accuracy (see above), it has a preference for one that is based on linear regression.

- Therefore, this exercise will include an evaluation of the suitability of using such a method.

6. At this stage, CEL is more interested in seeing evidence of the level of accuracy that the PoC can achieve rather than in what the user interface of any eventual production application might look like.
    - Therefore, provision of a prototype user interface is outside the scope of this exercise.

7. Because of the current challenging Brexit timelines, this PoC exercise must be completed no later than Friday, 17th May 2019.
8. Due to its limitiations in terms of timelines and budget, CEL authorises OAL to reuse code, where possible, and to maximise the use of automation during the development of the PoC (e.g. during training and testing of its resultant Machine Learning algoithm).

Approach

OAL used the following approach to complete this PoC:

1. Select the coding language and Integrated Development Environment (IDE).
2. Load the dataset provided by CEL into the selected IDE.
3. Analyse the dataset provided by CEL under the following headings:
    a. Missing data;
    b. Anomalous data;
    c. Presence of outliers;
    d. Presence of collinearity.
4. Address any limitations in the dataset identified during the previous step.
5. Assess the suitability of using a linear regression-based Machine Learning method.
6. Choose an alternative Machine Learning method (if found necessary during the previous step).
7. Select the fields (variables) that would optimise the predictive power of the chosen ML method (using a suitable automated stepwise selection approach with 5-fold cross validation).
8. Split the dataset into training and test datasets (using an 80:20 random split).
9. Train the selected Machine Learning algorithm using the data for the selected variables in the training dataset (also with 5-fold cross validation automation).
10. Test the selected Machine Learning algorithm using the data for the selected variables in the test dataset.
11. Document the test results (in terms of both performance and accuracy).

Findings

The following points summarise the results of this PoC exercise:

1. R was selected as the coding language and RStudio as Integrated Development Environment (IDE).
2. The dataset provided by CEL was successfully loaded into the selected IDE.
3. During the data analysis phase:
    a. Some of the property-related attributes were provided in the form of sets of two or more dummy variables which were consolidated into a series of single categorical variables with CEL's agreement.
    b. 10 records pertained to locations outside of the London Boroughs (spatial outliers) and were deleted from the dataset with CEL's agreement.
    c. The level of quality of the socio-economic data in the dataset was found to be very poor/suspect (both in terms of its consistency at the level of the dataset's areal units and due to presence of invalid values and the prevalence of zero values which was adjudged to be indicative of missing data). Therefore, that data was deleted from the dataset with CEL's agreement.
    d. Although the presence of outlier data was detected in the remaining data, it was considered not to be appropriate to delete the (potentially many) impacted records. Instead, selecting an outlier-insensitive (robust) Machine Learning algorithm was crucial.

    e. A significant level of multicollinearity (at the global level) was detected in the remaining data. This provided further impetus to select a Machine Learning method which would also be insensitive to it.
    f. An additional variable (field/column) was added to the loaded dataset which specified the price range interval into which the selling/purchase price of each property fell.
    g. One other variable was added to the loaded dataset which identified the London Borough in which each property is located (based on the values of its Easting & Northing cordinates).
4. In the light of findings 3d and 3e above a Machine Learning method based on Ordinary Least Squares (OLS) muliple linear regression was deemed not to be viable.
5. Due to finding 4 above, the size of the CEL dataset and its future growth potential (see requirement 1d above), and CEL's sub-second response time requirement (2), the use of a Geographically Weighted Regression-based method was also ruled out.
6. Because of the previous 2 findings, a Machine Learning method based on the K Nearest Neighbours (KNN) algorithm was chosen for this PoC (in consultation with CEL).
7. An automated stepwise variable selection process (based on the Random Forest algorthm) discovered that 11 of the 12 predictor variables could make a contribution toward optimising the predictive power of the selected Machine Learning method.
8. The dataset provided by CEL was successfully split into training and test datasets (using an 80:20 random split).
9. The selected Machine Learning algorithm KNN) was successfully trained using the data for the selected variables in the training dataset (with 5-fold cross validation automation).
10. The automated testing of the selected Machine Learning algorithm (KNN) using the data for the selected variables in the test dataset was completed successfully.
11. The results of that test prove that:
    a. An accuracy rate of around 70% is achievable.
    b. Sub-second response time is attainable.

Please refer to the Conclusions section at the end of this document for further details and to the intervening sections for information on how the various stages of this project were executed and what their results were.

Next Steps

Finally, we in OAL would like to thank CEL for the opportunity to develop this prototype and trust that it proves that the underlying technology has the potential to satisfy all of CEL's business requirments as set out that the beginning of this document. We look forward to discussing the next steps toward those ends with you in the near future.

Acknowledgments

Sean O'Riogain, the senior OAL staff member who developed this PoC, wishes to acknowledge the following:

1. The help and support of the following OAL colleagues in terms of peer reviewing the overall approach and much of its code and for supplying some of the code needed for generating maps of the London Boroughs:
    - Kathryn Dillon-Duffy;
    - Paul Williamson.
2. The reuse of some techniques and code from a similar type of engagement undertaken by OAL on behalf of another client, Brunson Electoral Research (BER), which was permitted under the terms of the contract under which that engagement was performed.
3. CEL's techical department provided R code for combining a set of dummy variables into a single factor and for producing a scatter plot that illustrates the link between purchase price and floor area.

Proof of Concept Details

The following subsections of this document show how each of the stages of this PoC (as outlined in the Approach section above) were executed and what their results were.

Data Loading & Wrangling

CEL's dataset was loaded and its structure reviewed as follows:

```
data1<-read.csv("DataScienceProj.csv")
str(data1)
```

```
## 'data.frame':    12536 obs. of  31 variables:
##  $ X       : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing: int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice: int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ BldIntWr: int  0 0 0 0 0 0 0 0 1 0 ...
##  $ BldPostW: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Bld60s  : int  1 0 0 0 0 0 0 0 0 1 ...
##  $ Bld70s  : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Bld80s  : int  0 1 0 1 1 0 0 0 0 0 ...
##  $ TypDetch: int  1 1 0 1 1 0 0 0 0 0 ...
##  $ TypSemiD: int  0 0 1 0 0 0 0 0 1 1 ...
##  $ TypFlat : int  0 0 0 0 0 1 1 1 0 0 ...
##  $ GarSingl: int  1 1 0 1 0 0 0 0 0 0 ...
##  $ GarDoubl: int  0 0 0 0 1 0 0 0 0 0 ...
##  $ Tenfree : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ BedTwo  : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ BedThree: int  1 1 0 1 0 1 0 0 1 1 ...
##  $ BedFour : int  0 0 1 0 1 0 1 0 0 0 ...
##  $ BedFive : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ NewPropD: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea: num  76.2 98.5 124.7 127 190.4 ...
##  $ NoCarHh : num  50.28 14.63 36.42 17.81 7.41 ...
##  $ CarspP  : num  25.2 46.9 37.8 47.6 67.7 ...
##  $ ProfPct : num  0 6.25 0 0 9.09 ...
##  $ UnskPct : num  11.1 0 11.1 0 0 ...
##  $ RetiPct : num  88.9 12.5 77.8 75 36.4 ...
##  $ Saleunem: num  19.23 5.36 5.26 8.82 3.68 ...
##  $ Unemploy: num  85.535 32.826 31.617 0.129 21.888 ...
##  $ PopnDnsy: num  11.49 8.29 7.82 18.18 8.22 ...
```

We can see from the structure of the data above that there are several groupings of dummy variables (starting with BldIntWr and ending with NewPropD) that we can consolidate to reduce the dataset's dimensionality - for the data analysis phase of this engagement at least.

Note that we will need to reinstate those dummy variables if the use of a linear regression-based Machine Learning method proves to be suitable.

This is because that type of ML method can only handle categorical variables if they are binary variables (aka dichotomous variables). That is, they can only take one of two possible values (e.g. Male or Female; On or Off, Up or Down; 0 or 1).**

Dummy Variables Replacement

The following code replaces each of the 4 non-binary sets of dummy variables with a single (factor) variable.

```
Dummy2Factor <- function(mat,lev1="Level1") {
      mat <- as.matrix(mat)
      factor((mat %*% (1:ncol(mat))) + 1,
          labels = c(lev1, colnames(mat)))
}

Bld <- Dummy2Factor(data1[,5:9],"PreWW1")            # Build (age)
Typ <- Dummy2Factor(data1[,10:12],"Others")          # Type
Gar <- Dummy2Factor(data1[,13:14],"HardStnd")        # Garage(s)?
Bed <- Dummy2Factor(data1[,18:21],"BedOne")          # Bedrooms

data <- data.frame(data1[,c(1:4,15:17,22,23:31)],Bld,Typ,Gar,Bed)

# Check that the expected results have been achieved
str(data)
```

```
## 'data.frame':    12536 obs. of  21 variables:
##  $ X       : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing: int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice: int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ Tenfree : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ NewPropD: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea: num  76.2 98.5 124.7 127 190.4 ...
##  $ NoCarHh : num  50.28 14.63 36.42 17.81 7.41 ...
##  $ CarspP  : num  25.2 46.9 37.8 47.6 67.7 ...
##  $ ProfPct : num  0 6.25 0 0 9.09 ...
##  $ UnskPct : num  11.1 0 11.1 0 0 ...
##  $ RetiPct : num  88.9 12.5 77.8 75 36.4 ...
##  $ Saleunem: num  19.23 5.36 5.26 8.82 3.68 ...
##  $ Unemploy: num  85.535 32.826 31.617 0.129 21.888 ...
##  $ PopnDnsy: num  11.49 8.29 7.82 18.18 8.22 ...
##  $ Bld     : Factor w/ 6 levels "PreWW1","BldIntWr",..: 4 6 1 6 6 1 5 1 2 4 ...
##  $ Typ     : Factor w/ 4 levels "Others","TypDetch",..: 2 2 3 2 2 4 4 4 3 3 ...
##  $ Gar     : Factor w/ 3 levels "HardStnd","GarSingl",..: 2 2 1 2 3 1 1 1 1 1 ...
##  $ Bed     : Factor w/ 5 levels "BedOne","BedTwo",..: 3 3 4 3 4 3 4 2 3 3 ...
```

The results above show that the relevant dummy variables have been replaced successfully.

Price Range Variable Addition

In accordance with requirement 1, the following code assigns a (categorical) price range value to each observation (property) based on its purchase price.

```
# This function provides the price range bracket for a give price
priceR<-function(x,k=50){                  # Set the default interval (k) at Â£50k
  if(k==0){                                # Handle a zero price
    r<-"NA"
  } else{                                  # For non-zero price
    y<-k*1000                              # Interval assumed in multiples of 1000
    r1<-((x%/%y)*y%/%1000)                 # Lower range value
    r2<-r1+(y/1000)                        # Upper range value
    r<-paste(r1,"-",r2,"k",sep="")         # Construct the range description
  }
  return(r)
}

data$PriceR<-factor(priceR(data$Purprice))  # Add the price range variable as a factor

str(data)                                  # Verify price range variable added
```

```
## 'data.frame':    12536 obs. of  22 variables:
##  $ X       : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing: int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice: int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ Tenfree : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ NewPropD: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea: num  76.2 98.5 124.7 127 190.4 ...
##  $ NoCarHh : num  50.28 14.63 36.42 17.81 7.41 ...
##  $ CarspP  : num  25.2 46.9 37.8 47.6 67.7 ...
##  $ ProfPct : num  0 6.25 0 0 9.09 ...
##  $ UnskPct : num  11.1 0 11.1 0 0 ...
##  $ RetiPct : num  88.9 12.5 77.8 75 36.4 ...
##  $ Saleunem: num  19.23 5.36 5.26 8.82 3.68 ...
##  $ Unemploy: num  85.535 32.826 31.617 0.129 21.888 ...
##  $ PopnDnsy: num  11.49 8.29 7.82 18.18 8.22 ...
##  $ Bld     : Factor w/ 6 levels "PreWW1","BldIntWr",..: 4 6 1 6 6 1 5 1 2 4 ...
##  $ Typ     : Factor w/ 4 levels "Others","TypDetch",..: 2 2 3 2 2 4 4 4 3 3 ...
##  $ Gar     : Factor w/ 3 levels "HardStnd","GarSingl",..: 2 2 1 2 3 1 1 1 1 1 ...
##  $ Bed     : Factor w/ 5 levels "BedOne","BedTwo",..: 3 3 4 3 4 3 4 2 3 3 ...
##  $ PriceR  : Factor w/ 12 levels "0-50k","100-150k",..: 10 10 10 10 5 1 1 10 1 10 ...
```

```
head(paste(data$X,": ",data$Purprice," (",  # Verify range assigned correctly
           data$PriceR,")",sep=""))
```

```
## [1] "53: 85000 (50-100k)"    "73: 71000 (50-100k)"
## [3] "78: 60000 (50-100k)"    "95: 64000 (50-100k)"
## [5] "125: 260000 (250-300k)" "153: 48500 (0-50k)"
```

As shown above, the new price range variable (PriceR) has been added to the dataset correctly.

London Borough Identifier Addition

The following code adds the London Borough identifier to each observation based on its Easting-Northing coordinate values.

```
# Create spatial data frame using the Northing and Easting variables
lh<-SpatialPointsDataFrame(cbind(data[,2:3]),data)

# Import the LondonBoroughs shape file data
lb<-readOGR(dsn=".", "LondonBoroughs", stringsAsFactors=F)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\oriogain\Dropbox\Maynooth\NCG613 - Data Analytics Project\Project - Solo", layer:
"LondonBoroughs"
## with 33 features
## It has 15 fields
## Integer64 fields read as strings:  NUMBER NUMBER0 POLYGON_ID UNIT_ID
```

```
class(lb)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
str(lb@data)
```

```
## 'data.frame':    33 obs. of  15 variables:
##  $ NAME      : chr  "Camden London Boro" "Tower Hamlets London Boro" "Islington London Boro" "Hackne
y London Boro" ...
##  $ AREA_CODE : chr  "LBO" "LBO" "LBO" "LBO" ...
##  $ DESCRIPTIO: chr  "London Borough" "London Borough" "London Borough" "London Borough" ...
##  $ FILE_NAME : chr  "GREATER_LONDON_AUTHORITY" "GREATER_LONDON_AUTHORITY" "GREATER_LONDON_AUTHORITY"
"GREATER_LONDON_AUTHORITY" ...
##  $ NUMBER    : chr  "77" "82" "84" "86" ...
##  $ NUMBER0   : chr  "1317" "1339" "1357" "1380" ...
##  $ POLYGON_ID: chr  "50632" "50746" "50581" "50673" ...
##  $ UNIT_ID   : chr  "11244" "11185" "11281" "11199" ...
##  $ CODE      : chr  "E09000007" "E09000030" "E09000019" "E09000012" ...
##  $ HECTARES  : num  2179 2158 1486 1905 2960 ...
##  $ AREA      : num  0 179 0 0 0 ...
##  $ TYPE_CODE : chr  "AA" "AA" "AA" "AA" ...
##  $ DESCRIPT0 : chr  "CIVIL ADMINISTRATION AREA" "CIVIL ADMINISTRATION AREA" "CIVIL ADMINISTRATION AR
EA" "CIVIL ADMINISTRATION AREA" ...
##  $ TYPE_COD0 : chr  NA NA NA NA ...
##  $ DESCRIPT1 : chr  NA NA NA NA ...
```

```
Bname<-gsub(" London Boro","",lb$NAME)               # Store the borough names
xy <- coordinates(lb)                                # Store the borough coordinates

proj4string(lh) <- CRS(proj4string(lb))              # Copy CRS

lhlb <- over(lh,lb)   # spatial join: points first, then polygons
dim(lhlb)
```

```
## [1] 12536    15
```

```
head(lhlb)           # data frame has lb attributes in lh order
```

```
##                                NAME AREA_CODE     DESCRIPTIO
## 1                 Bexley London Boro       LBO London Borough
## 2 Hammersmith and Fulham London Boro       LBO London Borough
## 3              Islington London Boro       LBO London Borough
## 4                Bromley London Boro       LBO London Borough
## 5                Croydon London Boro       LBO London Borough
## 6                 Merton London Boro       LBO London Borough
##                   FILE_NAME NUMBER NUMBER0 POLYGON_ID UNIT_ID      CODE
## 1 GREATER_LONDON_AUTHORITY     42    1080      50891   10759 E09000004
## 2 GREATER_LONDON_AUTHORITY     70    1254      50647   11259 E09000013
## 3 GREATER_LONDON_AUTHORITY     84    1357      50581   11281 E09000019
## 4 GREATER_LONDON_AUTHORITY      9     805      50904   10772 E09000006
## 5 GREATER_LONDON_AUTHORITY      6     781      51330   10896 E09000008
## 6 GREATER_LONDON_AUTHORITY     59    1213     122401   10995 E09000024
##    HECTARES    AREA TYPE_CODE                DESCRIPT0 TYPE_COD0
## 1  6428.649 371.119        AA CIVIL ADMINISTRATION AREA      <NA>
## 2  1715.409  75.648        AA CIVIL ADMINISTRATION AREA      <NA>
## 3  1485.664   0.000        AA CIVIL ADMINISTRATION AREA      <NA>
## 4 15013.487   0.000        AA CIVIL ADMINISTRATION AREA      <NA>
## 5  8649.441   0.000        AA CIVIL ADMINISTRATION AREA      <NA>
## 6  3762.466   0.000        AA CIVIL ADMINISTRATION AREA      <NA>
##   DESCRIPT1
## 1      <NA>
## 2      <NA>
## 3      <NA>
## 4      <NA>
## 5      <NA>
## 6      <NA>
```

```
str(lhlb)
```

```
## 'data.frame':    12536 obs. of  15 variables:
##  $ NAME      : chr  "Bexley London Boro" "Hammersmith and Fulham London Boro" "Islington London Bor
o" "Bromley London Boro" ...
##  $ AREA_CODE : chr  "LBO" "LBO" "LBO" "LBO" ...
##  $ DESCRIPTIO: chr  "London Borough" "London Borough" "London Borough" "London Borough" ...
##  $ FILE_NAME : chr  "GREATER_LONDON_AUTHORITY" "GREATER_LONDON_AUTHORITY" "GREATER_LONDON_AUTHORITY"
"GREATER_LONDON_AUTHORITY" ...
##  $ NUMBER    : chr  "42" "70" "84" "9" ...
##  $ NUMBER0   : chr  "1080" "1254" "1357" "805" ...
##  $ POLYGON_ID: chr  "50891" "50647" "50581" "50904" ...
##  $ UNIT_ID   : chr  "10759" "11259" "11281" "10772" ...
##  $ CODE      : chr  "E09000004" "E09000013" "E09000019" "E09000006" ...
##  $ HECTARES  : num  6429 1715 1486 15013 8649 ...
##  $ AREA      : num  371.1 75.6 0 0 0 ...
##  $ TYPE_CODE : chr  "AA" "AA" "AA" "AA" ...
##  $ DESCRIPT0 : chr  "CIVIL ADMINISTRATION AREA" "CIVIL ADMINISTRATION AREA" "CIVIL ADMINISTRATION AR
EA" "CIVIL ADMINISTRATION AREA" ...
##  $ TYPE_COD0 : chr  NA NA NA NA ...
##  $ DESCRIPT1 : chr  NA NA NA NA ...
```

```
# Append the London Borough Identifier to our original data frame
data$Boro<-gsub(" London Boro", "", x=lhlb$NAME)       # Strip spurious naming text
data$Boro<-factor(data$Boro)                           # Convert to a factor
str(data)                                              # Verify borough id appended
```

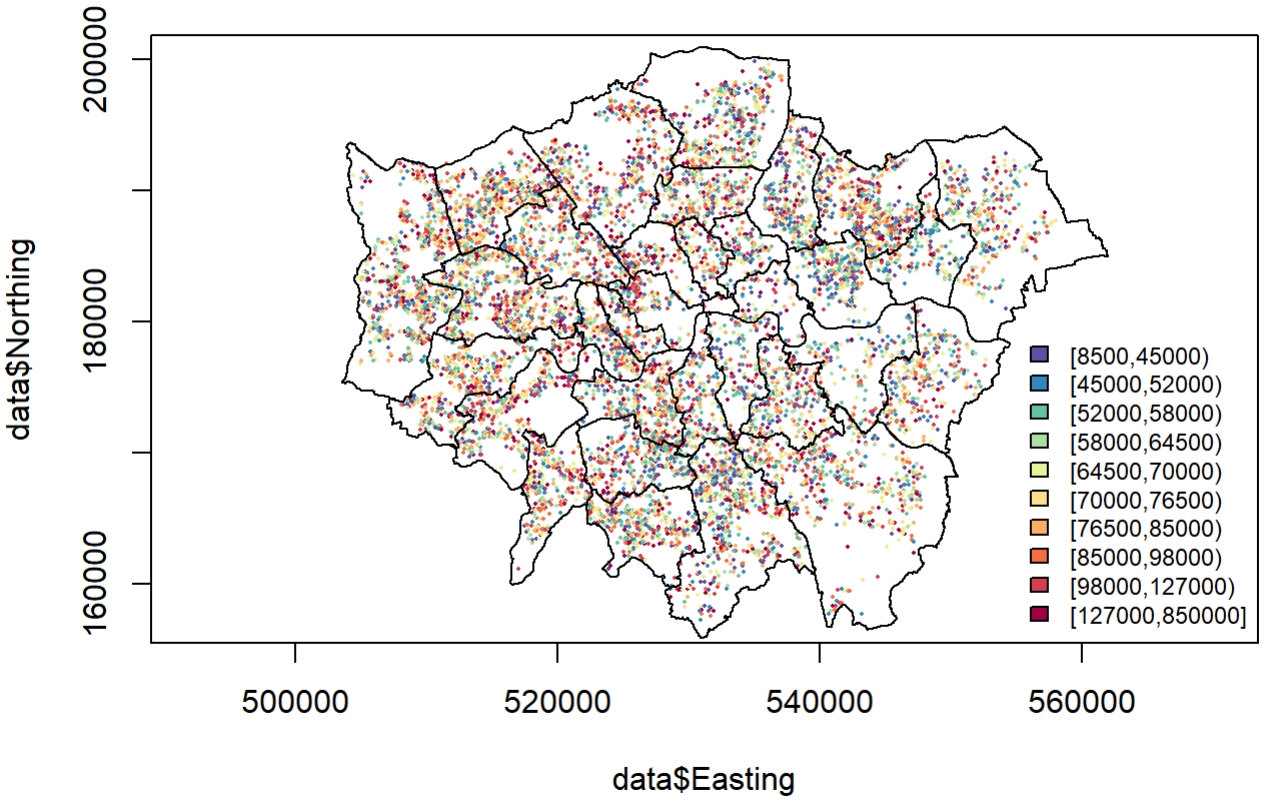
```
## 'data.frame':    12536 obs. of  23 variables:
##  $ X       : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing: int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice: int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ Tenfree : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ NewPropD: int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea: num  76.2 98.5 124.7 127 190.4 ...
##  $ NoCarHh : num  50.28 14.63 36.42 17.81 7.41 ...
##  $ CarspP  : num  25.2 46.9 37.8 47.6 67.7 ...
##  $ ProfPct : num  0 6.25 0 0 9.09 ...
##  $ UnskPct : num  11.1 0 11.1 0 0 ...
##  $ RetiPct : num  88.9 12.5 77.8 75 36.4 ...
##  $ Saleunem: num  19.23 5.36 5.26 8.82 3.68 ...
##  $ Unemploy: num  85.535 32.826 31.617 0.129 21.888 ...
##  $ PopnDnsy: num  11.49 8.29 7.82 18.18 8.22 ...
##  $ Bld     : Factor w/ 6 levels "PreWW1","BldIntWr",..: 4 6 1 6 6 1 5 1 2 4 ...
##  $ Typ     : Factor w/ 4 levels "Others","TypDetch",..: 2 2 3 2 2 4 4 4 3 3 ...
##  $ Gar     : Factor w/ 3 levels "HardStnd","GarSingl",..: 2 2 1 2 3 1 1 1 1 1 ...
##  $ Bed     : Factor w/ 5 levels "BedOne","BedTwo",..: 3 3 4 3 4 3 4 2 3 3 ...
##  $ PriceR  : Factor w/ 12 levels "0-50k","100-150k",..: 10 10 10 10 5 1 1 10 1 10 ...
##  $ Boro    : Factor w/ 33 levels "Barking and Dagenham",..: 3 14 20 5 9 25 13 5 24 16 ...
```

```
head(data$Boro)                                    # Verify text stripped
```

```
## [1] Bexley                Hammersmith and Fulham Islington
## [4] Bromley               Croydon                Merton
## 33 Levels: Barking and Dagenham Barnet Bexley Brent Bromley ... Wandsworth
```

Here is a map that provides an overview of CEL's territory, the London Boroughs, and the distribution of its sales across a 10-interval price range:

```
# Set the number of intervals
nClass = 10

# Choose the palette type, interval type (quantile) and use them to create the colour palette
Palette <- rev(brewer.pal(nClass,"Spectral"))
Classes <- classIntervals(data$Purprice,nClass,"quantile")
Colours <- findColours(Classes,Palette)

# Plot the sales locations and add the borough bounderies and legend
plot(data$Easting,data$Northing,pch=16,cex=0.25,col=Colours,asp=1)
plot(lb,add=TRUE)
legend("bottomright",
      legend=names(attr(Colours,"table")),
      fill=attr(Colours,"palette"),
      cex=0.75,bty="n")
```



There appears to be a higher concentration of the less expensive (blue/green) properties in the eastern half of the city relative to the western half. The opposite is the case for the more expensive (red/orange) properties.

We will now locate (in terms of row numbers) any instances where the spatial join (overlay) failed (because the relevant Easting-Northing coordinates do not fall within a London Borough, presumably).

```
which(is.na(lhlb$NAME)==T)
```

```
##  [1]   449  1549  3322  4855  6599  7432  9378 10833 10961 11569
```

The above results shows us that 10 such anomalies exist.

With CEL's agreement, we have elected to delete them from our dataset because they do not fall within its target geography - i.e. as defined by the London Borough boundaries.

```
# Delete rows with no Boro value
paste("Rows Before = ",
  nrow(data),sep="")                          # Number of rows in dataset
```

```
## [1] "Rows Before = 12536"
```

```
paste("For Deletion = ",
  nrow(filter(data, is.na(Boro))),sep="")       # Number of rows with no Boro value
```

```
## [1] "For Deletion = 10"
```

```
data<-filter(data, !is.na(Boro))               # Delete rows with no Boro value
paste("Rows After =  ",
  nrow(data),sep="")                           # Number of rows in dataset
```

```
## [1] "Rows After =  12526"
```

Factor to Numeric Variable Conversion

To allow for the possibility of using an alternative to linear regression for house price prediction, we can increase the pool of potential predictors by converting our new categorical variables to (pseudo) continuous ones using their level numbers instead of their values.

We can do so because nearly all of them are ordered (ordinal) variables (i.e. their level values reflect some form of magnitude) or they are boolean (e.g. Ten, Cen, New). For instance, the age of the property is quantified by the levels of the Bld variable and, number of bedrooms is quantified by the levels of the Bed variable and even its price range is quantifed by the levels of the PriceR variable. However, as well as applying this type of conversion to the PriceR and Boro variables we will also retain them in factor format to preserve their level descriptions for possible use later.

```
# Store the level values of the following factors:
PriceL<-levels(data$PriceR)
BoroL<-levels(data$Boro)

# Convert factors to integer variables
data$Bld<-as.integer(data$Bld)
data$Typ<-as.integer(data$Typ)
data$Gar<-as.integer(data$Gar)
data$Bed<-as.integer(data$Bed)
data$BoroNum<-as.integer(data$Boro)
data$Boro<-BoroL[data$Boro]               # Add the Boro level information variable
data$PriceRNum<-as.integer(data$PriceR)
data$PriceRTxt<-PriceL[data$PriceRNum]    # Add the Price Range level info variable

str(data)                                 # Check the results
```

```
## 'data.frame':    12526 obs. of  26 variables:
##  $ X        : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting  : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing : int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice : int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ Tenfree  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat  : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo  : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ NewPropD : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea : num  76.2 98.5 124.7 127 190.4 ...
##  $ NoCarHh  : num  50.28 14.63 36.42 17.81 7.41 ...
##  $ CarspP   : num  25.2 46.9 37.8 47.6 67.7 ...
##  $ ProfPct  : num  0 6.25 0 0 9.09 ...
##  $ UnskPct  : num  11.1 0 11.1 0 0 ...
##  $ RetiPct  : num  88.9 12.5 77.8 75 36.4 ...
##  $ Saleunem : num  19.23 5.36 5.26 8.82 3.68 ...
##  $ Unemploy : num  85.535 32.826 31.617 0.129 21.888 ...
##  $ PopnDnsy : num  11.49 8.29 7.82 18.18 8.22 ...
##  $ Bld      : int  4 6 1 6 6 1 5 1 2 4 ...
##  $ Typ      : int  2 2 3 2 2 4 4 4 3 3 ...
##  $ Gar      : int  2 2 1 2 3 1 1 1 1 1 ...
##  $ Bed      : int  3 3 4 3 4 3 4 2 3 3 ...
##  $ PriceR   : Factor w/ 12 levels "0-50k","100-150k",..: 10 10 10 10 5 1 1 10 1 10 ...
##  $ Boro     : chr  "Bexley" "Hammersmith and Fulham" "Islington" "Bromley" ...
##  $ BoroNum  : int  3 14 20 5 9 25 13 5 24 16 ...
##  $ PriceRNum: int  10 10 10 10 5 1 1 10 1 10 ...
##  $ PriceRTxt: chr  "50-100k" "50-100k" "50-100k" "50-100k" ...
```

The results above show that the required variable conversions and additions have been completed successfully.

Note that by doing these conversions now, we can also include them in our outlier and correlation analysis studies later.

Range of Values Review

The following subsections decribe how each of the variables in the original dataset provided by CEL were assessed for basic data quality in terms of their value ranges.

Unique Identifier

The uniqueness of the values of variable X, which seems to be the dataset's unique identifier, was checked as follows:

```
paste("X values are unique? =",      # Check that this is the dataset's unique id
      length(unique(data$X))==length(data$X))
```

```
## [1] "X values are unique? = TRUE"
```

The result of the code above indicates that the value of the X variable can be used to uniquely identify each observation in the dataset.

Northing and Easting

The spatial join that we have done previously implicitly ensures the validity of the easting-northing coordinate value pairs vis-a-vis their location within one of the London boroughs.

We will now take a look at the level of resolution of the Easting and Northing variables by looking at the level of uniqueness (in terms of coordinates) that they achieve across the dataset.

```
# Calculate the percentage uniqueness
coords_uni<-length(unique(paste(data$Easting,",",data$Northing,sep="")))
coords_tot<-length(paste(data$Easting,",",data$Northing,sep=""))
paste("Unique Coordinates Percentage: ", round(coords_uni/coords_tot*100,2),
      "% (", coords_uni, " of ", coords_tot, ")", sep="")
```

```
## [1] "Unique Coordinates Percentage: 75.66% (9477 of 12526)"
```

```
# Check if this has been caused by deliberate location obfuscation (by settiing the 2 rightmost digits
 of the Easting & Northing variables to zeros, apparently)
sum(data$Easting %% 100==0)
```

```
## [1] 12526
```

```
sum(data$Northing %% 100==0)
```

```
## [1] 12526
```

As can be seen from the calculations above, only about 75% of the properties have unique Easting-Northing coordinate pairs. As it appears that all Easting and Northing values are exactly divisible by 100 (for location obfuscation purposes, presumably), we cannot treat this lack of uniqueness as a potential data quality issue.

Purchase Price

The following code checks that the values of this variable fall within a reasonable range. For instance, observations with very low values (e.g. zero or close to it) would require further investigation.

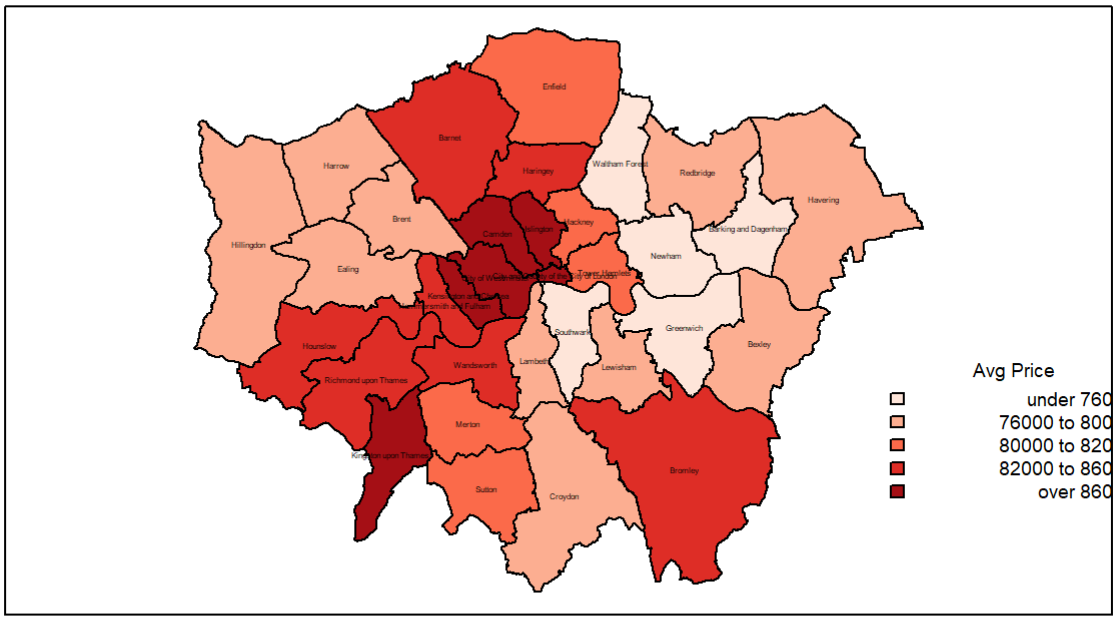
```
summary(data$Purprice)    # Check the minimum and maximum values versus their median
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8500   55000   70000   80025   90000  850000
```

The above results show that, while no price is very close to zero the fact that the minimum is as low as Â£8,500 and the maximum is at Â£850,000, when compared with a median value of Â£70,000, indicates that there is a high degree of variability in this variable which will need to be investigated further during outlier analysis.

Here is a choropleth map of the London Boroughs showing the spatial distribution of average purchase price:

```
# Get the average purchase price by borough
SQL_avgp<-sqldf("select Boro, AVG(Purprice) as avgp
                 from data
                 group by boro;")

# Set up the map shading for 5 cuts
p<-match_order(SQL_avgp$Boro, Bname)
shades<-auto.shading(SQL_avgp[p,,drop=FALSE]$avgp,
                     cutter=quantileCuts,n=5,cols=brewer.pal(5, "Reds"))

# Render the choropleth map
choropleth(lb, SQL_avgp[p,, drop=FALSE]$avgp,
           shading=shades)

# Add the borough names
text(xy[,1],xy[,2],Bname,col="black",cex=0.25)

# Add the legend for 5 cuts
choro.legend(558000, 175000,shades,title='Avg Price',cex=.6, bty="n",
             fmt="%0.0f")

# Add a border
box()
```



This map clearly shows how the most expensive properties spread out north and south from the city centre as well as the Bromley borough in the southeast.

Floor Area

As per the purchase price, we need to take a high-level look at the range of values for this variable.

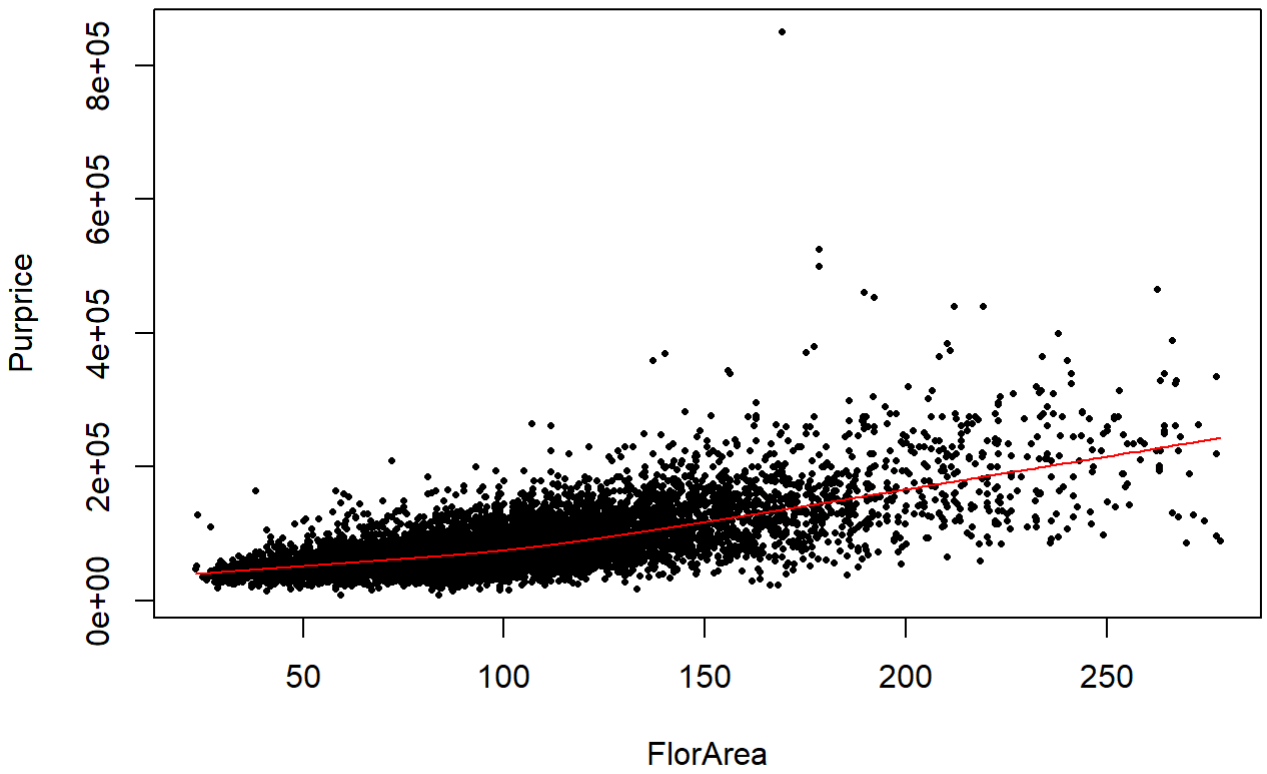
```
summary(data$FlorArea)    # Check the minimum and maximum values versus their median
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   23.22   71.98   91.02   96.50  112.11  278.00
```

The metadata for this dataset informs us that this variable is specified in units of square meters which means that above results tell us that it ranges from about $23m^2$ (c. 250 square feet) to $278m^2$ (c. 3,000 square feet) with a median value of $91m^2$ (c. 1,000 square feet). While the median value appears to be reasonable enough, the relatively low values at the lower and upper ends of the range demand further scrutiny during outlier analysis.

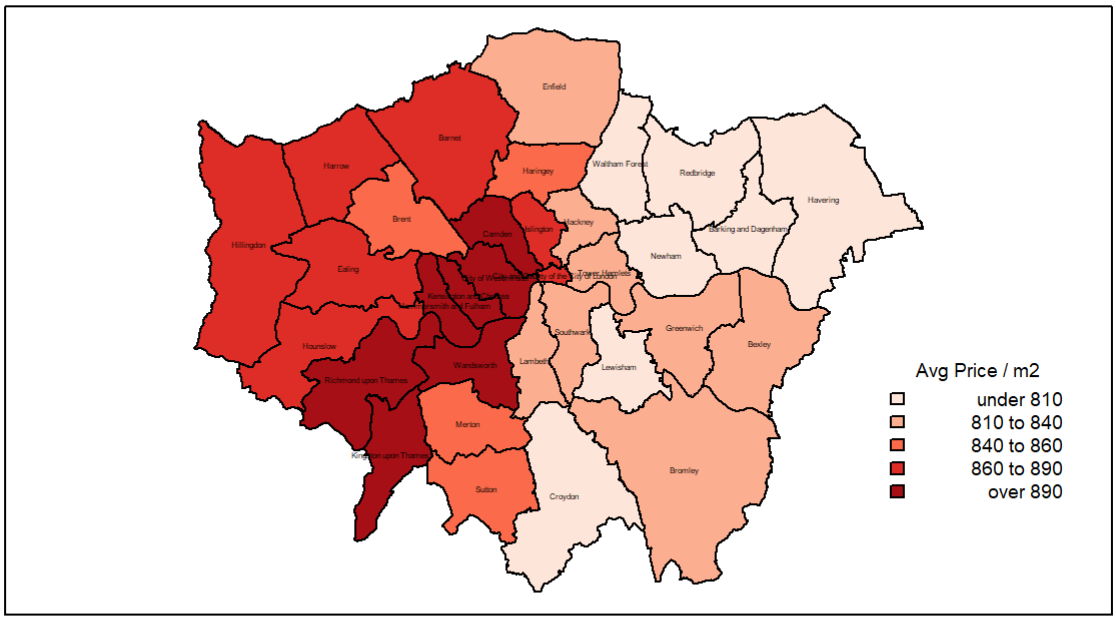
The following plot shows how, in general, purchase price rises in line with floor area:

```
plot(data[,c("FlorArea","Purprice")],pch=16,cex=0.5)
lines(lowess(data[,c("FlorArea","Purprice")]),col="red")
```



Here is a choropleth map of the London Boroughs showing the spatial distribution of average purchase price per square meter:

```
# Get the average purchase price by borough
SQL_avgpm2<-sqldf("select Boro, AVG(Purprice/FlorArea) as avgpm2
                  from data
                  group by boro;")

# Set up the map shading for 5 cuts
p<-match_order(SQL_avgpm2$Boro, Bname)
shades<-auto.shading(SQL_avgpm2[p,,drop=FALSE]$avgpm2,
                     cutter=quantileCuts,n=5,cols=brewer.pal(5, "Reds"))

# Render the choropleth map
choropleth(lb, SQL_avgpm2[p,, drop=FALSE]$avgpm2,
           shading=shades)

# Add the borough names
text(xy[,1],xy[,2],Bname,col="black",cex=0.25)

# Add the legend for 5 cuts
choro.legend(558000, 175000,shades,title='Avg Price / m2',cex=.6, bty="n",
             fmt="%0.0f")

# Add a border
box()
```



This map shows a slightly different picture of the location of the most expensive properties per square meter with a clear east-west divide.

Dummy Variables

With the removal of the most of groupings of dummy variables, data analysis on them is no longer required or possible.

Socio-Economic Indicator Variables

The metadata for this dataset shows that there are number of variables that provide various socio-economic indicators which can be used to assess the level of deprivation (or otherwise) that prevails at the location of each property and which, in turn, could influence the house prices in that locality. The variables in question are (mostly) expressed as proportions/percentanges of the local population, the validity of whose values are checked in the following subsections.

Consistency Analysis

We will firstly take a look at these variables to ensure that they take the same values at the same location (at coordinate level). Even though, as we have seen earlier, those coordinates have been modified to obfuscate actual location, properties with the same obfuscated coordinates should be in close proximity to each other in actuality. Therefore, we would expect to see those properties (or a sizeable proportion of them, at least) to have the same values for all of the socio-economic variables.

For starters, let's check that there are properties with the same (obfuscated) coordinate pairs:

```
sqldf("select easting, northing, count(*) as count
      from data
      group by easting, northing
      having count(*) > 1
      order by 3 desc
      limit 10;")
```

```
##    Easting Northing count
## 1   520600   188700    17
## 2   531100   183400    15
## 3   534600   191700    12
## 4   511600   181900     9
## 5   521800   178800     9
## 6   535700   180300     9
## 7   507600   181900     8
## 8   509300   178400     7
## 9   511900   181900     7
## 10  512300   182500     7
```

As it happens, the results above can show that there are 2,186 easting-northing coordinate pairs at which more than one property is located. The results of that query have been limited to show the top 10 in the table above. It tells us, for instance, that there are 17 properties with the easting-northing coordinate values (520600, 188700).

Now let's use that query to look at the values of the socio-economic variables for a sample of 20 properties which have those same (sample) coordinate value pairs.

```
sqldf("select d.easting, d.northing, d.nocarhh, d.carspp, d.profpct, d.unskpct,
      d.retipct, d.saleunem, d.unemploy, d.popndnsy
      from data d
      inner join (
        select easting, northing, count(*) as count
        from data
        group by easting, northing
        having count(*) > 1
        order by 1, 2
        limit 10
      ) X on X.easting = d.easting and X.northing = d.northing
      order by 1, 2
      limit 20;")
```

```
##    Easting Northing NoCarHh  CarspP ProfPct UnskPct  RetiPct Saleunem
## 1   504400   191100 10.8108 68.2353  0.0000  0.0000 200.0000   2.7027
## 2   504400   191100 32.6180 37.9798  7.1429  0.0000  42.8571   7.6412
## 3   504400   191100 41.6149 29.7436  7.6923  7.6923  15.3846  15.5340
## 4   504700   183300  5.8824 63.9506 10.0000  0.0000  30.0000   2.4752
## 5   504700   183300 14.1553 47.2628  6.2500  6.2500  25.0000   5.2265
## 6   505000   183200 37.8238 40.2913  0.0000  8.3333  41.6667   2.7132
## 7   505000   183200 17.4359 40.1852 30.0000 10.0000  50.0000  11.7647
## 8   505000   188200 36.9792 39.5062 14.2857  7.1429  28.5714   5.6034
## 9   505000   188200 18.7817 58.7054 55.5556  0.0000  55.5556   5.5794
## 10  505000   188900 56.7568 22.1007  0.0000 10.0000  60.0000  20.5263
## 11  505000   188900 62.3037 19.4323  0.0000  0.0000  66.6667  23.6715
## 12  505000   188900 49.2064 17.3034  0.0000 12.5000  37.5000  21.0256
## 13  505200   181600 31.2000 33.0658  5.8824  0.0000  29.4118  11.3043
## 14  505200   181600 16.6667 53.5047 66.6667  0.0000 633.3333  15.6863
## 15  505200   183200 35.2174 37.9496  6.6667 13.3333  40.0000   6.6421
## 16  505200   183200 49.7207 23.0469  0.0000  0.0000  28.5714  17.5879
## 17  505300   181500 40.4494 40.4891 11.1111  0.0000  77.7778   7.4074
## 18  505300   181500 21.6418 51.7483 16.6667  0.0000 116.6667   5.2632
## 19  505300   181500 15.9420 47.0363  0.0000  7.6923  46.1539   4.9834
## 20  505300   184600 29.1005 42.2917  0.0000  0.0000  30.0000  10.8949
##     Unemploy PopnDnsy
## 1    0.22883  2.73973
## 2   89.33713  8.73606
## 3   87.57021 10.63291
## 4    1.47243  7.52427
## 5   15.71694  7.77778
## 6   26.16223 11.74652
## 7   90.35889  8.78243
## 8   46.82245  9.20578
## 9   68.96320 11.07078
## 10  52.31007 13.48039
## 11 111.23490  4.63710
## 12 195.99640 11.25000
## 13  20.10230  8.43882
## 14  29.20053  4.91803
## 15  69.83131  5.31401
## 16  39.72187  6.98925
## 17 139.10820  9.03361
## 18  22.04864  1.53846
## 19  39.45007  6.01770
## 20  46.67273  4.86381
```

We can see from even a 20-instance subset of the results of the query above that properties in the same locality (i.e. with the same (obfuscated) coordinates) have wildly differing values for their socio-economic variables.

As an aside, we can also see that individual properties have the same value (down to 4 decimal places) for more than one variable. For instance, one of the properties at (504400, 191100) has the value 7.6923 for both the percentage of professional and unskilled workers (how likely is that?), and one of the properties at (505000, 188200) has the value 55.5556 for the both the percentage of professional workers and retirees (even if we ignore the fact that their sum is greater than 100, how likely is that?).

These observations (sic) provide us with an initial impression that leads us to the conclusion that the socio-economic data subset is meaningless and, therefore, should not be used by this proof-of-concept exercise. However, we will refrain from removing them until our data quality review of them has been completed at least.

Value Range Analysis

Let’s take a high-level look at the range of that these variables exhibit:

```
apply(data[,10:17], 2, summary)
```

```
##          NoCarHh   CarspP   ProfPct   UnskPct  RetiPct  Saleunem  Unemploy
## Min.     0.00000  0.00000   0.00000  0.000000   0.0000  0.000000   0.00000
## 1st Qu. 14.81480 31.83557   0.00000  0.000000  18.7500  4.713800  15.06589
## Median  26.96355 41.60015   5.55560  0.000000  35.2941  7.142900  38.85643
## Mean    29.26743 40.89632   7.63975  4.216562  45.9548  8.924694  47.17665
## 3rd Qu. 41.26210 50.39370  12.50000  7.692300  58.3333 11.354900  64.88040
## Max.    92.61750 84.31370 100.00000 71.428600 900.0000 90.000000 686.98930
##          PopnDnsy
## Min.     0.000000
## 1st Qu.  5.295478
## Median   7.622910
## Mean     8.881738
## 3rd Qu. 10.869560
## Max.    82.802550
```

From the results above, we have identified the following potential issues that will required further investigation:

1. The minumum value of all of them is zero which may be indicative of missing values (for instance, a population density of zero does not make sense in a populous city like London).
2. The median value of the UnskPct variable is zero which may be flagging to us that there are a lot of missing values for it.
3. The maximum values of both the RetiPct and Unemploy variables indicate the presence of invalid (percentage) values.

Missing Values Analysis

```
miss<-function(x){sum(x==0)}            # Function to count zero value
table(apply(data[,10:17],1,miss))       # Count zero values for all variables
```

```
##
##    0    1    2    3    4    5    6    8
## 1784 6648 3709  305   67    7    5    1
```

```
table(apply(data[,10:11],1,miss))       # Count zero values for car-related vars
```

```
##
##     0     1     2
## 12487    34     5
```

```
table(apply(data[,12:14],1,miss))       # Count zero values for job-related vars
```

```
##
##    0    1    2    3
## 1803 6735 3735  253
```

```
sum(data[,13]==0)                       # Count zero values for Unsk$Pct var
```

```
## [1] 8129
```

```
table(apply(data[,15:16],1,miss))       # Count zero values for unemployment vars
```

```
##
##     0     1     2
## 12350   172     4
```

```
sum(data[,17]==0)                       # Count zero values for PopnDnsy var
```

```
## [1] 134
```

The main points arising from the results above are as follows:

1. Only 1784 of the remaining 12,526 observations have no zeros in any of these variables.
2. One observation has all zeros for these variables.
3. Most observations (12,487) have non-zero values for both private transport-related variables (i.e. NoCarHh and CarspP).
4. Only 1,803 of the observations have non-zero values for all of occupation-related variables (i.e. ProfPct, UnskPct and RetiPct),
5. As expected, the value of the UnskPct variable is zero for a large number (8,129) of the observations.
6. Most observations (12,350) have non-zero values for both unemployment-related variables (i.e. Saleunem and Unemploy, assuming that the former is unemployment-related).
7. 134 observations have zero values for the population density (PopnDnsy) variable.

Now let's take a look at the minimum values of these variables if we discount their zero values:

```
min_nonzero<-function(x){               # Function to get min of non-zero values
  min(subset(x,x!=0))
}

apply(data[10:17],2,min_nonzero)
```

```
##  NoCarHh   CarspP  ProfPct  UnskPct  RetiPct Saleunem Unemploy PopnDnsy
##  0.44840  2.29890  3.84620  3.22580  2.56410  0.41490  0.01713  0.15267
```

The above results lead us to the following conclusion:

- They all look like reasonable percentage values except that the non-zero minimum value of the Unemploy variable appears to be implausibly low (but we will be taking a closer look at that variable later).

Invalid Data Analysis

As noted previously, the RetiPct and Unemploy appear to have invalid percentage values so the first step is to quantify the scale of the problem as follows:

```
# Display counts of the invalid values (we are treating 100 as potentially invalid)
sqldf("select count(*) as Bad_RetiPct from data where RetiPct>=100;")
```

```
##   Bad_RetiPct
## 1        1208
```

```
sqldf("select count(*) as Bad_Unemploy from data where Unemploy>=100;")
```

```
##   Bad_Unemploy
## 1         1177
```

```
# Display counts of sample instances of the invalid values
sqldf("select RetiPct, count(*) as count from data where RetiPct>=100 group by RetiPct order by 2 desc
 limit 10;")
```

```
##     RetiPct count
## 1  100.0000   343
## 2  133.3333    57
## 3  125.0000    52
## 4  150.0000    51
## 5  112.5000    48
## 6  114.2857    47
## 7  128.5714    41
## 8  166.6667    40
## 9  200.0000    38
## 10 111.1111    36
```

```
sqldf("select Unemploy, count(*) as count from data where Unemploy>=100 group by Unemploy order by 2 de
sc limit 10;")
```

```
##    Unemploy count
## 1  149.7309     6
## 2  110.6984     5
## 3  101.1620     4
## 4  195.9964     4
## 5  298.3107     4
## 6  100.5397     3
## 7  101.0593     3
## 8  104.9848     3
## 9  105.8593     3
## 10 108.0757     3
```

```
# Display the minimum values
summary(data[,c(14,16)])
```

```
##     RetiPct          Unemploy
##  Min.   :  0.00   Min.   :  0.00
##  1st Qu.: 18.75   1st Qu.: 15.07
##  Median : 35.29   Median : 38.86
##  Mean   : 45.95   Mean   : 47.18
##  3rd Qu.: 58.33   3rd Qu.: 64.88
##  Max.   :900.00   Max.   :686.99
```

We can see from the results above that around 10% of the values of both variables are invalid.

One possible explanation for those invalid values is that they might have been erroneously multiplied by 100. Dividing them all by 100 would give reasonable maximum values (while dividing by 10 would still give them very big but valid values). However, this would make their mimimum values implausibly low. Therefore, we have concluded that the values of this variable have been corrupted in some way and that they should be removed from the dataset.

However, as all of our analysis of the socio-economic data subset to this point has done nothing to assuage our initial concerns as to its overall level of validity (as expressed in the Consistency Analysis section above), we have decided to remove all of the variables from the dataset (with CEL's agreement) on the grounds that they are likely to have a deleterious effect on the results of any house price predictions that relied upon them.

```
data<-data[,-c(10:17)]        # Drop the Soci-Economic variables
str(data)
```

```
## 'data.frame':    12526 obs. of  18 variables:
##  $ X        : int  53 73 78 95 125 153 182 189 203 207 ...
##  $ Easting  : int  545500 525000 531100 538500 534000 528700 534900 537700 534700 514400 ...
##  $ Northing : int  173000 177800 183400 169400 168400 168800 187000 169700 171600 188600 ...
##  $ Purprice : int  85000 71000 60000 64000 260000 48500 34500 55995 45000 60000 ...
##  $ Tenfree  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ CenHeat  : int  1 1 1 1 1 1 1 1 0 1 ...
##  $ BathTwo  : int  0 0 1 0 1 0 0 0 0 0 ...
##  $ NewPropD : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ FlorArea : num  76.2 98.5 124.7 127 190.4 ...
##  $ Bld      : int  4 6 1 6 6 1 5 1 2 4 ...
##  $ Typ      : int  2 2 3 2 2 4 4 4 3 3 ...
##  $ Gar      : int  2 2 1 2 3 1 1 1 1 1 ...
##  $ Bed      : int  3 3 4 3 4 3 4 2 3 3 ...
##  $ PriceR   : Factor w/ 12 levels "0-50k","100-150k",..: 10 10 10 10 5 1 1 10 1 10 ...
##  $ Boro     : chr  "Bexley" "Hammersmith and Fulham" "Islington" "Bromley" ...
##  $ BoroNum  : int  3 14 20 5 9 25 13 5 24 16 ...
##  $ PriceRNum: int  10 10 10 10 5 1 1 10 1 10 ...
##  $ PriceRTxt: chr  "50-100k" "50-100k" "50-100k" "50-100k" ...
```

The results above confirm the the socio-economic variables have been successfully removed from the dataset.

Outlier Analysis

Outlier analysis is not appropriate for the first 3 columns because the first one is just a unique identifier and the other two are coordinates which have already been implicitly checked for (spatial) outliers by the previous map plotting exercise.

Because we have not yet performed any variable (or model) selection, we will have to take a univariate approach to outlier detection (as opposed to a multivariate one).

As we now have numeric representations of all of the potential predictors, they are in scope for this exercise.

The objective of this exercise is not to definitively identify all genuine outliers based on the weight of evidence provided by a number of detection methods. It is rather to get a high-level impression (at the global level) of the preponderance (or otherwise) of outlier values in the univariate context.

To do this, we will write a function to count the outliers per variable using the same definition of an outlier (by John Tukey) that is used in box plots as follows:

```
# This function uses the same definition of an outlier (by John Tukey) that is used in box plots
out.count<-function(x){
  min<-summary(x)[2]*1.5       # 1.5 times the 1st quantile
  max<-summary(x)[5]*1.5       # 1.5 times the 3rd quantile

  return(sum(x<min | x>max))   # Return the count
}

apply(data[,4:13],2,out.count)
```

```
## Purprice  Tenfree  CenHeat  BathTwo NewPropD FlorArea      Bld      Typ
##     9479        0    12526      676      455     9529     6324     3787
##      Gar      Bed
##     8300     5706
```

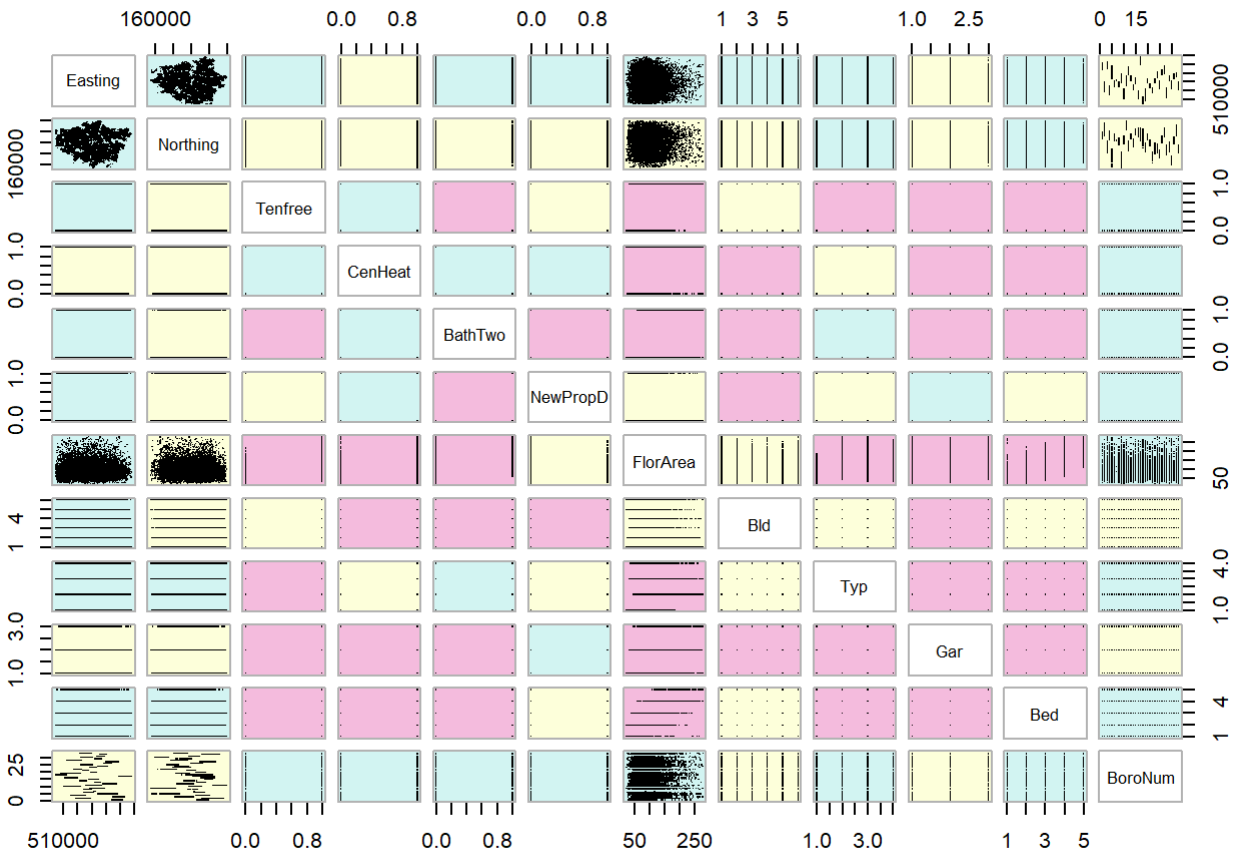
While the outlier counts for the erstwhile dummy variables (especially the binary ones) may not have much significance, there is a clear indication of the presence of outliers in what is effectively the dataset's response variable (Purprice) and in what turns out to be its main predictor variable (FlorArea).

Of course, the existence of outliers is a fact of life in all areas of human endeavour and buying and selling property is no exception. For instance, it is not unheard of for the buyer of a property to get a bargain (despite the vendor's or agent's best efforts). Similarly, it is not uncommon for a vendor to achieve an exceptionlly high price for his property (e.g. in the event of a 'bidding war'). As this is a phenomenon of the marketplace, we cannot simply choose to delete observations containing outliers and, therefore, we need to use a Machine Learning method that is insensitive to their presence (i.e. a robust method).

Correlation Analysis

The objective of this exercise is to get an overview of the presence (or otherwise) and the strength of collinearity in our dataset *at the global level only*.

```
data.cor<-cor(data[,c(2:3,5:13,16)])
data.cor %>%
  dmat.color()->cols
cpairs(data[,c(2:3,5:13,16)],panel.colors=cols,pch=".",gap=.5)
```



The cpairs variant of the standard scatter plot matrix (pair) provides a kind of heat map of the level of correlation exists between all variable pairings in the data set. It does this by cutting the correlation values of the variable pairings into 3 equal intervals (breaks): red for noteworthy positive correlation, cream for noteworthy negative correlation and blue for weak or no correlation.

As can be seen from its output above, there is evidence of quite a bit of multicollinearity in the dataset under review, which also applies to its spatial (Easting-Northing coordinate) variables.

The following code will display the value intervals for the cream, blue and red colour codes above:

```
cormat<-cor(data[,c(2:3,5:13,16)])                  # Create the correlation matrix

n<-nrow(cormat)                                     # Get number of rows

for(i in c(1:n)){                                   # 'Blank' the redundant cells
  for(j in c(i:n)){
    cormat[i,j]<-NA
  }
}

cut<-cut_number(cormat,3)                           # Cut cormat into 3 ranges
levels(cut)                                         # Display the intervals/breaks
```

```
## [1] "[-0.154,-0.0061]" "(-0.0061,0.0583]" "(0.0583,0.803]"
```

The intervals above indicate that, where strong correlation exists, it is positive (with a Pearson r value of up to 0.803).

We will now take a closer look at the noteworthy correlations (both positive and negative) by producing a correlation matrix with a view to quantifying it in descending order of importance (as measured by their Pearson r values).

```
cormat<-cor(data[,c(2:3,5:13,16)])                  # Create the correlation matrix

n<-nrow(cormat)                                     # Get number of rows

for(i in c(1:n)){                                   # 'Blank' the redundant cells
  for(j in c(i:n)){
    cormat[i,j]<-NA
  }
}

cut<-cut_number(cormat,3)                           # Cut cormat into 3 ranges

corind<-which(as.numeric(cut) %in% c(1,3))          # Find cells with highest cor value

corrow<-corind %% n                                 # Convert cell index to row no.
corcol<-corind %/% n + 1                            # Convert cell index to col. no.

fix<-which(corrow==0)                               # corrow/corcol elements to fix
corrow[fix]<-n                                      # Fix corrow
corcol[fix]<-corcol[fix]-1                          # Fix corcol

rowname<-attr(cormat,"dimnames")[[1]][corrow]       # Convert row number to name
colname<-attr(cormat,"dimnames")[[2]][corcol]       # Convert column number to name
val<-cormat[corind]                                 # Get r value

cordf<-data.frame(rowname,colname,val)              # Create correlation data frame

head(arrange(cordf,desc(abs(val))),10)              # Display correlation data frame
```

```
##     rowname  colname       val
## 1       Typ  Tenfree 0.8031059
## 2       Bed FlorArea 0.7494574
## 3       Bed  Tenfree 0.6028859
## 4  FlorArea  Tenfree 0.4925491
## 5       Bed      Typ 0.4534975
## 6       Bld NewPropD 0.4100169
## 7  FlorArea  BathTwo 0.3819696
## 8       Bed      Gar 0.3272399
## 9       Gar FlorArea 0.3224441
## 10      Typ FlorArea 0.3095775
```

```
                                                    # (largest absolute r values first)
```

The table above shows (for the top ten pairings only) that there is a significant amount of strong multicollinearity involving what would appear be important potential predictors (e.g. floor area, number of bedrooms, property type and type of tenure).

Machine Learning Algorithm Selection

The results of the study documented in the Outlier Analysis and Collinearity Analysis sections above lead us to seek a Machine Learning method other than one based on Linear Regression (which are sensitive to both types of data).

Therefore, we will now proceed to use K Nearest Neighbours (KNN) because it is more robust on both fronts. However, there are 2 key problems to solve for this method to be successful:

1. Selection of the correct predictors.
2. Choosing the optimum number of nearest neighbours (k) to use.

Fortunately, the caret package in R provides us with the following functionality to solve both of those problems:

1. Recursive Feature Selection (RFE) which also comes with inbuilt cross validation for problem 1.
2. A model training function (called train) which also comes with inbuilt cross validation for problem 2.

That functionality will be implemented in the following sections of document.

Of course one of the other advantages of using KNN in place of linear regression is that it can be used for classification as well as regression. This means that we can use it to directly predict a price range instead of predicting an exact price and then deriving its price range interval. A potential disadvantage of KNN is that how it arrived at its predictions is not interpretable - but, as this is not a requirement for this project, that does not pose a problem for us.

Training-Test Set Split

The following code splits (based on a 80:20 ratio) the complete dataset into a training set and a test set.

```
set.seed(123)                                 # Set the seed value for repeatability
s<-sample(seq(to=nrow(data)), nrow(data)*0.8) # Get a random sample of 80% of obs.

train<-droplevels(data[s,])                   # Create the training set using sample
test<-droplevels(data[-s,])                   # Create the test set from the rest

train$PriceR<-factor(priceR(train$Purprice))  # Regenerate the Price Range factors
test$PriceR<-factor(priceR(test$Purprice))    #   after the split

dim(train)                                    # Verify the training set size
```

```
## [1] 10020    18
```

```
dim(test)                                    # Verify the test set size
```

```
## [1] 2506   18
```

The row counts of the training and test sets above indicate that the split has been performed as expected.

Predictor Selection using Recursive Feature Elimination (with Random Forest)

Even though we are not now going to use linear regression for modelling (prediction) purposes, before we proceed with using RFE, let's fit a linear regression model for all predictors and look at the p-values for the predictors in the resultant fit which it considers to be the most significant variables statistically.

```
# Fit a model with all (12) predictors
f.lm <- lm(PriceRNum ~ .,data = data[,c(2:3,5:13,16:17)])
summary(f.lm)
```

```
##
## Call:
## lm(formula = PriceRNum ~ ., data = data[, c(2:3, 5:13, 16:17)])
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.556 -4.295  1.941  2.806  8.684
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.165e+01  1.662e+00   7.012 2.48e-12 ***
## Easting     -7.913e-06  2.823e-06  -2.803  0.00507 **
## Northing    -3.120e-06  3.802e-06  -0.821  0.41189
## Tenfree      3.364e-01  1.445e-01   2.329  0.01988 *
## CenHeat      7.870e-01  1.055e-01   7.461 9.15e-14 ***
## BathTwo     -1.230e+00  1.657e-01  -7.424 1.21e-13 ***
## NewPropD    -6.527e-02  2.004e-01  -0.326  0.74470
## FlorArea    -3.146e-02  1.492e-03 -21.093  < 2e-16 ***
## Bld         -7.810e-02  2.387e-02  -3.271  0.00107 **
## Typ          3.738e-01  4.862e-02   7.689 1.59e-14 ***
## Gar          4.480e-02  7.344e-02   0.610  0.54183
## Bed          6.254e-01  6.329e-02   9.882  < 2e-16 ***
## BoroNum     -2.716e-03  3.706e-03  -0.733  0.46361
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.802 on 12513 degrees of freedom
## Multiple R-squared:  0.07316,    Adjusted R-squared:  0.07227
## F-statistic: 82.31 on 12 and 12513 DF,  p-value: < 2.2e-16
```

The results above indicate that almost all of the predictors have some degree of significant linear relationship with selling/purchase price (range). Surprisingly, of those that don't, two of them are location-related - i.e. the Northing coordinate and the Boro (number).

The following code now sets up RFE to perform its work across a number of cores in parallel (as it is very heavy on resources) and configures it to use Random Forest with 5-fold cross validation for all potential predictors in the training set.

Note that RFE inexplicably fails when the response variable is a factor, so we had to use its numeric equivalent (PriceRNum) instead as a work-around.

```
# This function builds a formula (f) involving the named predictors (x) and the specified response vari
able name (y)
form<-function(x,y){          # Initialise the formula using the response variable name
  f<-paste(y,"~",x[1],sep="") #  and that of the name of the first predictor
  for(i in 2:length(x)){      # Use a loop to append the remaining predictor names
    f<-paste(f,"+",x[i],sep="")
  }
  return(as.formula(f))       # Create (classify) the formula and return it
}

######################

subsetSizes <- c(1:12)               # Specify the number of predictors

# Set the seed value to predictably set the sample seed values for the cluster helpers
set.seed(123)

# Set a separate seed value for each of the 5 folds in a list
seeds <- vector(mode = "list", length = 6)
for(i in 1:5) seeds[[i]] <- sample.int(1000, length(subsetSizes) + 1)
seeds[[6]] <- sample.int(1000, 1)

# Set the debug file name (so that we can monitor the progress of this long-running process)
sys_date<-Sys.Date()
outfile<-paste("debug-rfe-",as.character(sys_date),".txt",sep="")

# Create (register) a parallel processing cluster to use all available cores except 1 (for the operatin
g system)
cl <- makeCluster(detectCores() - 1, type='PSOCK', outfile=outfile)
registerDoParallel(cl)

# Define the control using a random forest selection function with 5-fold CV
control <- rfeControl(functions=rfFuncs,    # Random Forest
                      method="cv",          # with cross validation
                      seeds=seeds,          # using the seed values generated above
                      verbose=TRUE,         # with logging to the debug file set on
                      number=5)             # for this number of folds

#### Note that we have to use the numeric form (PriceRNum) of the predictor instead of its factor versi
on (PriceR) which would have been referable (even after trying to build and rebuild the factor in diffe
rent ways). This is because we get the following error message when we try to use PriceR in the formul
a: "Can't have empty classes in y". RFE or RF appears to be indicating that there are some levels with
 no value. But the results of the following command contradicts that:

table(train$PriceR)
```

```
##
##    0-50k 100-150k 150-200k 200-250k 250-300k 300-350k 350-400k 400-450k
##     1586     1295      388      134       60       18        7        2
## 450-500k  50-100k 850-900k
##        3     6526        1
```

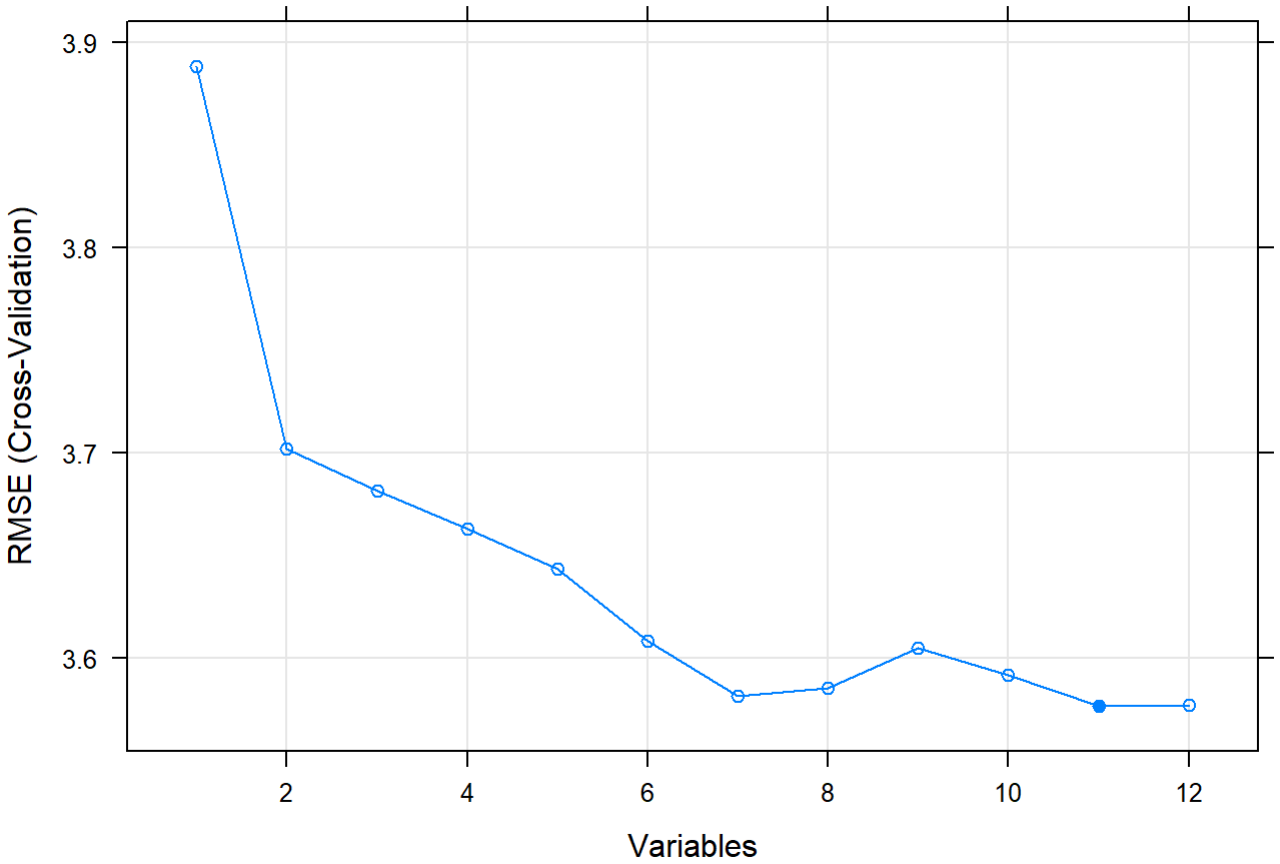
```
# Create the formula for all predictor variables
f<-form(names(train[,c(2:3,5:13,16)]), "PriceRNum")

# Invoke RFE (as configured by its control variable above) to produce a model that selects what Random
 Forest considers to be the optimal set of predictors
rf_model <- rfe(form=f, data=train,
               sizes=subsetSizes,rfeControl=control)

# Deregister the cluster
stopCluster(cl)

# Summarize the results
print(rf_model)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (5 fold)
##
## Resampling performance over subset size:
##
##  Variables  RMSE Rsquared   MAE  RMSESD RsquaredSD   MAESD Selected
##          1 3.888  0.07812 3.160 0.02225   0.005137 0.01794
##          2 3.702  0.12704 3.318 0.01288   0.008083 0.04645
##          3 3.682  0.13752 3.288 0.01983   0.014305 0.04467
##          4 3.663  0.14596 3.264 0.02782   0.015523 0.02302
##          5 3.643  0.15823 3.246 0.02352   0.013461 0.01324
##          6 3.608  0.16325 3.019 0.03708   0.014644 0.03040
##          7 3.582  0.17504 2.996 0.04509   0.017240 0.04010
##          8 3.586  0.17339 2.998 0.04537   0.017659 0.04071
##          9 3.605  0.16732 2.969 0.04249   0.015663 0.04031
##         10 3.592  0.17212 2.967 0.04886   0.018487 0.04678
##         11 3.576  0.17824 2.963 0.04372   0.016840 0.04157        *
##         12 3.577  0.17936 2.941 0.05093   0.020266 0.04813
##
## The top 5 variables (out of 11):
##    FlorArea, Tenfree, Typ, Bed, Bld
```

```
# List the chosen features
predictors(rf_model)
```

```
##  [1] "FlorArea" "Tenfree"  "Typ"      "Bed"      "Bld"      "Easting"
##  [7] "Northing" "BoroNum"  "BathTwo"  "Gar"      "CenHeat"
```

```
# Plot the results
plot(rf_model, type=c("g", "o"))
```



The results above show us that Random Forest is telling us that 11 of the 12 predictors can contribute to towards prediction accuracy based on their RMSE values. Therefore, we will only them when building (training) our KNN model in the following section of this document.

Coincidentally (or otherwise), NewPropD is the only predictor it didn't recommend and that was also the least significant variable (i.e. the one with the highest p-value) as per our earlier linear regression model fitting.

Interestingly, its graph also show that add the location-based predictors (Easting, Northing and Boro), which it ranks in positions 6, 7 and 8, make a relatively modest contribution to the predictive power of the model.

These results are broadly in agreement with those achieved by fitting a multiple linear regression model for all of our predictors.

Note that we also tried using just the first 7 predictors (not documented here) because the RMSE value for them is very close to the lowest one achieved overall but that resulted in a slightly worse success rate.

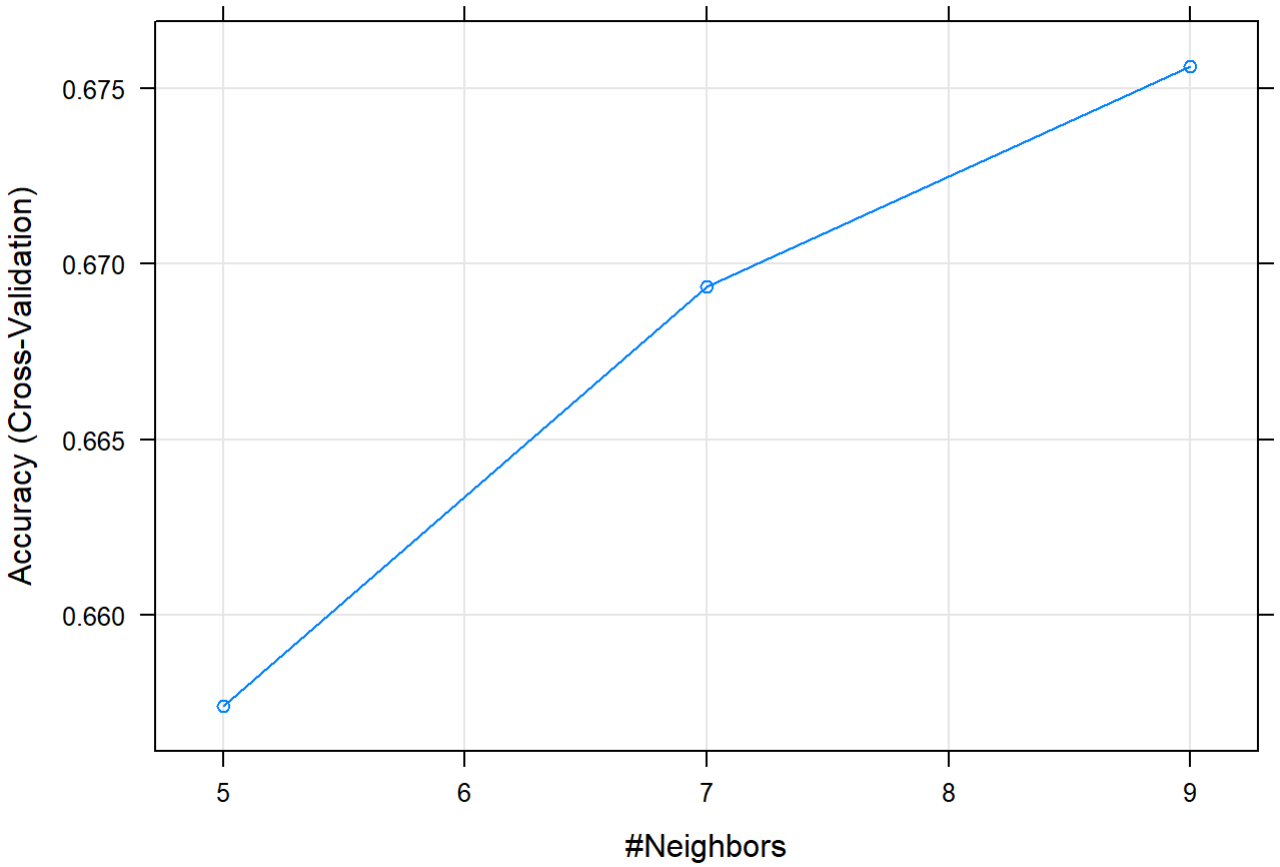
Machine Learning Algorithm (KNN) Training

The following code uses the train function in the caret package to automate the testing of our KNN algorithm. It uses a configurable cross validation process to automate the selection of the number of nearest neighbours (k) to create the best predictive model for the specified set of predictor variables (the 11 predictors recommended by RFE/RF above).

```
# Create the formula for the KNN model using all of the predictors recommended by RFE (i.e. all of them except New). But this time we can use the factorised Price Range as the response variable without a problem.
f<-form(names(train[,c(2:3,5:7,9:13,16)]), "PriceR")

#
knn_fit<-train(f,data=train,                    #
      trControl=trainControl(method="cv",       # Configure cross validation
                             number=5, allowParallel = FALSE),
      preProcess = c("center","scale"),         # Standardise the predictor values
      method="knn")                             # Use the KNN algorithm

# Display the details of the resultant model
knn_fit
```

```
## k-Nearest Neighbors
##
## 10020 samples
##    11 predictor
##    11 classes: '0-50k', '100-150k', '150-200k', '200-250k', '250-300k', '300-350k', '350-400k', '400-450k', '450-500k', '50-100k', '850-900k'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 8017, 8015, 8016, 8016, 8016

## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.6573839  0.2770095
##   7  0.6693619  0.2856891
##   9  0.6756488  0.2867788
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
# Visualise how the value of k was selected
plot(knn_fit)
```




```
# Test the model on the training set
pred_train<-predict(knn_fit)

# Calculate the success rate
rate_train<-round(sum(pred_train==train$PriceR)/length(train$PriceR),2)

# Display the success rate
paste("Training Accuracy:", rate_train)
```

```
## [1] "Training Accuracy: 0.73"
```

The results above that our KNN model was created successfully for which the optimum value of k=9 was selected.

Although its success rate when used with the training set is not brilliant, it does satisfy the relevant requirement as set out at the beginning of this document.

Let's hope that this models predictive power is sufficient to achieve a similar rate on the test set…….

Machine Learning Algorithm (KNN) Testing

We will now use the KNN model created during the model training phase above to perform prediction on the test set and calculate the success rate achieved using data that was previously 'unseen' by the model.

```
# Perform a prediction on the test set
pred_test <- predict(knn_fit, newdata = test)

# Calculate the success rate
rate_test<-round(sum(as.character(pred_test)==as.character(test$PriceR))/
                   length(test$PriceR),2)

# Display the accuracy rate
paste("Test Accuracy:", rate_test)
```

```
## [1] "Test Accuracy: 0.67"
```

The results above indicate that our new selling/purchase price (range) prediction model achieved an accuracy rate of 69% on the test set.

Conclusions

While the average of the success rates achieved by this model only barely achieves the 70% rate set out at the beginning of this document, OAL is very confident that, with more time, it can be tuned to attain even greater levels of accuracy and performance and to do so for even narrower price intervals.

This can be approached on a number of fronts including:

1. Reviewing the predictor selection.
2. Exploring the myriad of tuning parameters that both RFE/RF and KNN offer
    - but which, unfortunately, we have been unable to do within the tight timescales that govern this project.
3. Trying Random Forest for prediction as well as variable selection.

On the plus side, the model produced predictions for the both the training and test sets (with c. 10,000 and 2,500 records, respectively) within a few seconds in total so satisfying CEL's sub-second response time requirement for individual predictions will not be a problem.

Furthermore, should this initiative proceed to the development phase, this R prototype will very likely be replaced by a production version that will most likely be developed in Python (subject to CEL's agreement). In that case, Python offers even better performance and configurability - e.g. Manhattan (City Block) distance calculation which might prove to be more effective than that Euclidian distance, the only option currently offered by the KNN inplementation in R's caret library.