R Session 1 - Public Houses per 1,000 population

Michael Sinclair

4 October 2018

# Introduction to R - Session 1

In this introductory session we are going to be loading a comma seperated value text file (csv) in to R, carrying out a simple calculation and saving the data back to a new csv file.

We will be using base R for all of today’s exercise. We will look into using libraries in the next session.

We will be taking the following steps;

* Load data in to R
* Carry out simple calculation
* Use a very simple plot to explore the data
* Save dataframe back to csv file

# Working directory

It is possible to work in R without doing this by providing the full path to each file that you intend to work with. This can however increase the amount of code that you need to use and can also lead to files accidentally being saved in the wrong folder and potentially getting lost.

To set the working directory we will use the *setwd( )* command.

# You can either use a forward slash ( / ) or a double backslash ( \\ )  
  
setwd('//dsfin/corpcom/LBBDR/TNG/Session 1')

To check the active directory that you are working in you can use the *getwd( )* command. This will return the current working directory.

getwd()

## [1] "\\\\dsfin/corpcom/LBBDR/TNG/Session 1"

# Load data in to R

We will use the *read.csv( )* command to load data in to R from a csv file.

## Assignment operator

When we load the data in to R we will assign it to a variable which we will call *df* as the data type we will be using is a **dataframe**.

In R there are two wasy of doing this. The standard way is to use the *<-* symbol. This is the method that is widely used and the short cut to place this symbol is **ALT** + **-**.

R will also accept the *=* symbol as is used in other programming languages.

## Data Frame

A DataFrame is a special tabular data type, similar to a spreadsheet, which can be used to store data in rows and columns. Each column in a dataframe can hold a different type of data such as characters, numerical data and boolean (True / False).

df <- read.csv('session1.csv')

### Head()

Once you have read in the data use the *head( )* command to explore the top rows of the dataframe. By default this shows the top 6 rows, however you can provide an additional arguement to the head() function to provide more or less.

head(df) #You can also use tail(df) to view the bottom of the dataframe

## area ph2006 ph2016 population2006 population2016  
## 1 City of London 185 150 7239 9401  
## 2 Barking and Dagenham 40 20 167060 206460  
## 3 Barnet 120 100 330729 386083  
## 4 Bexley 105 90 223217 244760  
## 5 Brent 100 75 276534 328254  
## 6 Bromley 120 100 301048 326889

## Working with dataframes

There are two ways to work with a dataframe these are using *names* or *indexes* for the rows and columns.

In order to select the first column from the data you can select by name by using the *$* sign as follows.

df$area

## [1] City of London Barking and Dagenham Barnet   
## [4] Bexley Brent Bromley   
## [7] Camden Croydon Ealing   
## [10] Enfield Greenwich Hackney   
## [13] Hammersmith and Fulham Haringey Harrow   
## [16] Havering Hillingdon Hounslow   
## [19] Islington Kensington and Chelsea Kingston upon Thames   
## [22] Lambeth Lewisham Merton   
## [25] Newham Redbridge Richmond upon Thames   
## [28] Southwark Sutton Tower Hamlets   
## [31] Waltham Forest Wandsworth Westminster   
## 33 Levels: Barking and Dagenham Barnet Bexley Brent Bromley ... Westminster

Or you can use the column index, in this case the index is 1. Enclose this in square brackets as follows.

Use a single set of square brackets to access the column of data *[1]* and double square brackates to access the actual figures *[[1]]*.

df[[1]]

## [1] City of London Barking and Dagenham Barnet   
## [4] Bexley Brent Bromley   
## [7] Camden Croydon Ealing   
## [10] Enfield Greenwich Hackney   
## [13] Hammersmith and Fulham Haringey Harrow   
## [16] Havering Hillingdon Hounslow   
## [19] Islington Kensington and Chelsea Kingston upon Thames   
## [22] Lambeth Lewisham Merton   
## [25] Newham Redbridge Richmond upon Thames   
## [28] Southwark Sutton Tower Hamlets   
## [31] Waltham Forest Wandsworth Westminster   
## 33 Levels: Barking and Dagenham Barnet Bexley Brent Bromley ... Westminster

# Basic calculations

In today’s example we will be carrying out a calculation using two columns from the dataframe to caculate the number of public houses per London borough per 1,000 population.

To do this we will use the following calculation

We will carryout this calculation and save it as a new column within the dataframe *(df)*.

df$PHper1000\_2006 <- (df$ph2006 / df$population2006) \* 1000  
df$PHper1000\_2016 <- (df$ph2016 / df$population2016) \* 1000  
  
head(df) #use head() to see the two new columns

## area ph2006 ph2016 population2006 population2016  
## 1 City of London 185 150 7239 9401  
## 2 Barking and Dagenham 40 20 167060 206460  
## 3 Barnet 120 100 330729 386083  
## 4 Bexley 105 90 223217 244760  
## 5 Brent 100 75 276534 328254  
## 6 Bromley 120 100 301048 326889  
## PHper1000\_2006 PHper1000\_2016  
## 1 25.5560160 15.95574939  
## 2 0.2394349 0.09687106  
## 3 0.3628348 0.25901166  
## 4 0.4703943 0.36770714  
## 5 0.3616192 0.22848160  
## 6 0.3986075 0.30591424

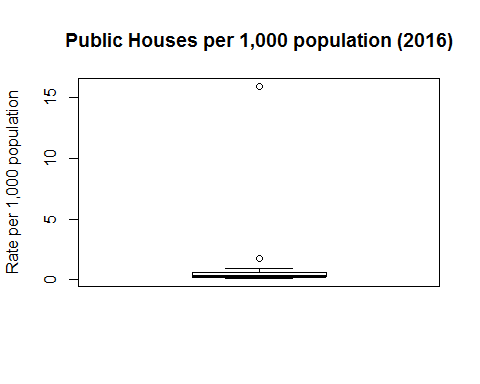
# Some basic plots

## Distribution

A boxplot is a very good way to look at the distribution of one or more numberical datasets.

A box plot will show us the distribution of the data and help us to identify whether we have any outliers.

boxplot(df$PHper1000\_2016, main='Public Houses per 1,000 population (2016)', ylab='Rate per 1,000 population')



In this example there is an extreme outlier, City of London, who have a very small population of 9401 and 150 public houses.

We can remove this outlier by removing the first row of the dataframe.

When selecting columns and rows from a dataframe use the square bracket method in a dataframe[row, column] format.

If you leave either the row value or coumn value blank you are selecting **all** rows or columns.

If you use a minus *(-)* symbol you are select all rows / columns but the one(s) you mention.

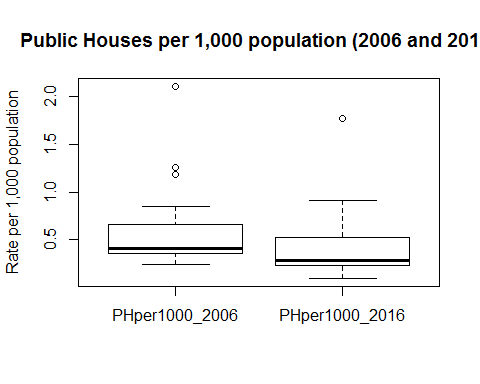
Therefore we remove the first row as follows. Don’t forget to assign the filtered dataframe back to the varaible df.

df <- df[-1, ]

Display the boxplot again to see if it has improved.

We can also pass more than one column into the boxplot function to compare two years. The following chart compares 2006 and 2016 data.

boxplot(df[6:7], main='Public Houses per 1,000 population (2006 and 2016)', ylab='Rate per 1,000 population')



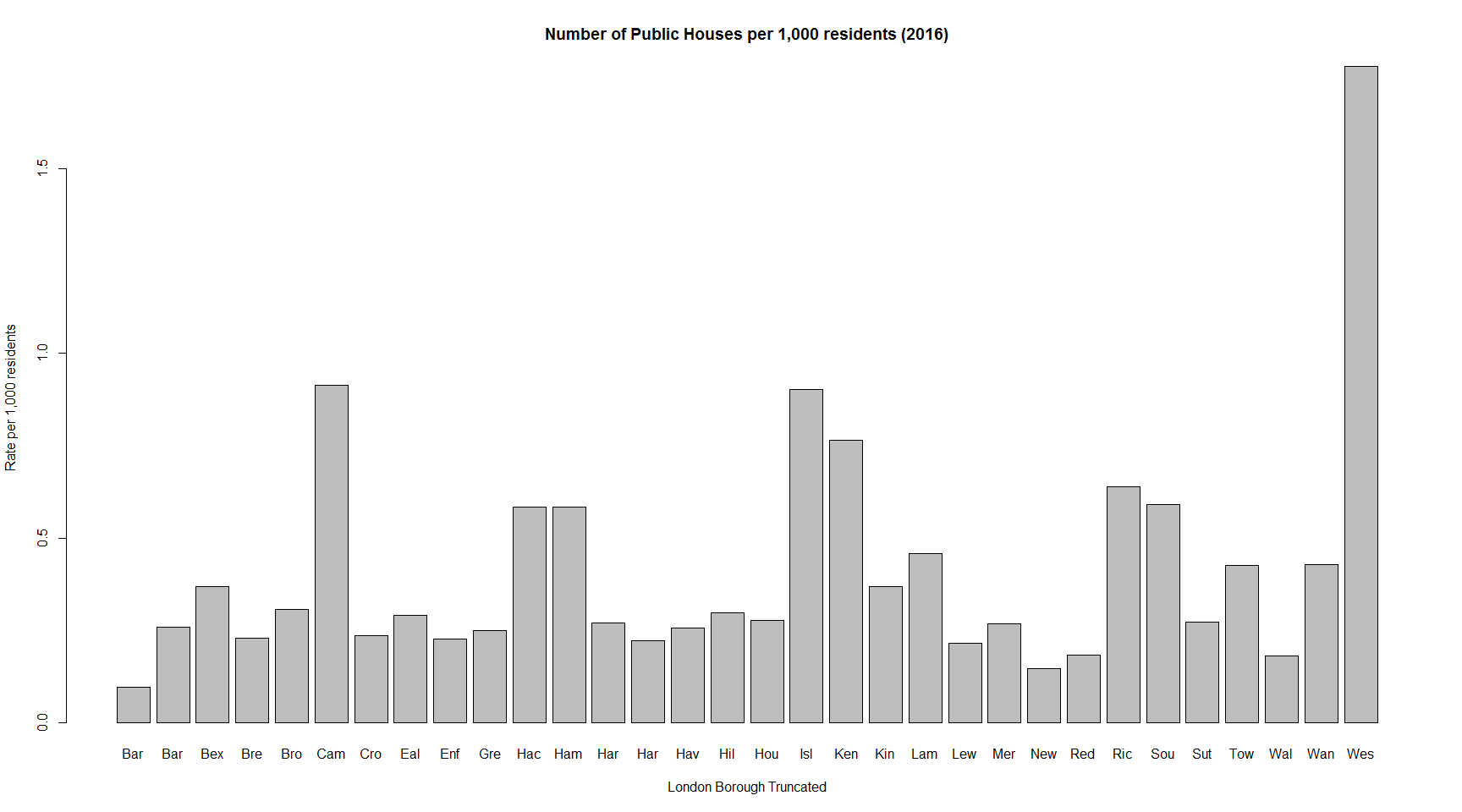
We can see that there is still a single outlier (Westminster) however depending on our analysis we may want to keep that is the data.

## Bar chart

You may want to see a bar plot of the data for each London borough.

In today’s session we will do this displaying the boroughs in alphabetical order. In future sessions we will explore sorting the data before plotting.

barplot(height = df$PHper1000\_2016, names.arg = substr(df$area, 1, 3), main = 'Number of Public Houses per 1,000 residents (2016)', xlab='London Borough Truncated', ylab='Rate per 1,000 residents')



# Save data to csv

Now we have created new data we can save it to a new csv file. In this case the processing has not been very difficult but it is standard practice to keep your original data unmodified and to then keep a *‘clean’* version of the data and possibly a *modelled* or *processed* version also.

To do this we will use the *write.csv( )* command. This command requires two arguements the data to be written and the file is to be written to.

By default R will include the row names in the first column of the saved csv file. In order to prvent this provide the additional arguement *row.names = FALSE* when writing the csv file.

Also note that R will over write any file that you instruct it to without a warning. *Please be careful!*

write.csv(df, 'session 1 data with rate.csv', row.names = FALSE)