London fire station analysis in R

In this exercise we use R and R-Studio to explore a dataset to help inform policy.

To begin, you will need a copy of rStudio you can use Amazon AWS to launch the latest version using the AMIs here. Note this will cost you on a per hour basis. A number of pre-loaded instances may be available on the training course.

The sections of R Studio

Top right - Your environment: here you will see a list of datasets and variables you create or load.

Bottom right - Outputs and explorer: here you will see plots, results and other outputs from functions you call.

Bottom left - The console: where you type commands.

Step 1 - Load some data

From the environment panel (top right) click import dataset and select the “from URL” option.

When prompted enter the URL of our dataset

http://bit.ly/LF_Analysis

Name the dataset **lfa**

Once loaded enter the following in the console to default to using this dataset:

```
> attach(lfa)
```

This tells R to look in this dataset by default when doing plots and other commands. R allows you to load as many datasets as you like simultaneously, so you have to define which dataset in every command (unless you use the attach function).

Step 2 - Explore the data

Enter the following in the console to start exploring the data:

```
> str(lfa)
```

This will display the structure of the dataset. At this stage you should check the number of London Boroughs there are in the dataset. You can do this by checking the number of levels in the data for boroughs. Are there 33 entries (32 boroughs and City of London)? If not we might have errors in our input data.

To find out more about any command type a '?' followed by that command. For example, why not try:

```
> ?str
```

We can quickly find out some summary stats using the summary command:

```
> summary(lfa)
```

This will give us numerical or text summaries of each column based upon the structure.

Step 3 - London performance

The dataset used here gives details of all incidents attended by the London Fire Brigade since the 7th January 2013, the date that 10 fire stations were closed in London.

Prior to these closures, the mean response time in London was 5m 34s (334 seconds).

Find out what the mean response time is after the closures from the new dataset.

```
> mean(firstpumparriving_attendancetime)
```

Did you get N/A as a response?

This is because there are blank values in the dataset; we need to explicitly tell R to ignore them.

```
> mean(firstpumparriving_attendancetime, na.rm = TRUE)
```

This is the mean impact over the whole of London. So what was the impact of the closures?

Using an adaption of the mean command you can also find the mean response time for each borough. You can press the UP ARROW to get the previous command back and edit it.

```
> mean(firstpumparriving_attendancetime[incgeo_boroughname=="westminster"], na.rm = TRUE)
```

Step 4 - Plotting the data (boxplots)

We can use the many plot functions in R to examine the data further.

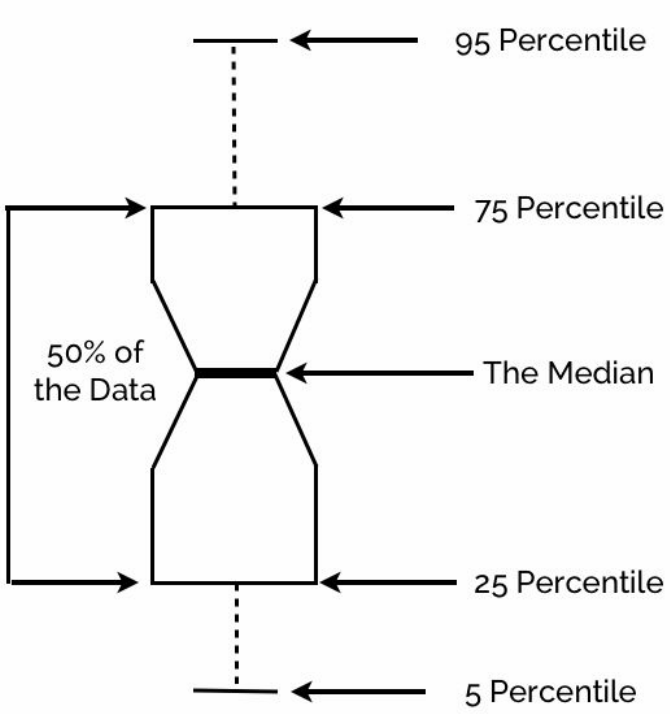
Let's start with a boxplot in order to look at the distribution of all response times.

```
> boxplot(firstpumparriving_attendancetime)
```

The plot can be customised to be horizontal and as a more traditional boxplot.

```
> bboxplot(firstpumparriving_attendancetime, horizontal = TRUE, axes = FALSE, staplewex = 1);
```

The boxplot shows us some key statistics about this dataset, including in what range 50% of the data is contained within. This is our InterQuartile Range (IQR). The diagram below helps explain the rest of the plotted data. Any dots outside of the 90% (5-95 percentiles) are considered outliers.



To see the data we can add labels to it.

```
> text(x=fivenum(firstpumparriving_attendancetime), labels
=fivenum(firstpumparriving_attendancetime), y=1.3, cex = 0.7, srt = 45)
```

Here *y* is the position from the axis, *cex* is the font size and *srt* is the label rotation (in degrees).

All of these figures are currently in seconds. We can load and use a library to parse these seconds into human readable times. NOTE: This library takes time to install.

```
> install.packages('lubridate')
```

```
> library('lubridate')
```

Once loaded you will need to redraw the boxplot. You can do this through pressing the up arrow until you find the command that starts with boxplot, then press enter to re-execute it.

To add the labels with human readable times press up to find the text command and change it to the following.

```
> text(x=fivenum(firstpumparriving_attendancetime), labels =
seconds_to_period(fivenum(firstpumparriving_attendancetime)), y=1.3, cex =
0.7, srt = 45)
```

You should now be able to easily see the distribution of the attendance time data.

Using similar functionality from before it is also possible to show a boxplot for a certain borough. In this case, Croydon.

```
boxplot(firstpumparriving_attendancetime[incgeo_boroughname=="croydon"],
horizontal = TRUE, axes = FALSE, staplewex = 1);
```

Each time you re-plot the boxplot, the text will be removed. Be careful as you can add lots of text layers by running the text command more than once.

```
text(x=fivenum(firstpumparriving_attendancetime[incgeo_boroughname=="croydon"]
), labels =
seconds_to_period(fivenum(firstpumparriving_attendancetime[incgeo_boroughname=
="croydon"])), y=1.3, cex = 0.7, srt = 45)
```

Step 5 - Histograms

Another way of looking at the distribution of the data is to use a histogram. There is a built in *hist* function in R.

```
> hist(firstpumparriving_attendancetime)
```

This histogram shows the distribution of the data overall. The built in hist function is fairly basic, so we are going to load a library to give us a more powerful and flexible histogram function.

```
> library(lattice)
```

Plot the same histogram with this library.

You can also display a histogram for a specific borough:

```
> histogram(firstpumparriving_attendancetime[incgeo_boroughname=="croydon"])
```

Next we can add a line which represents the mean:

```
> histogram(firstpumparriving_attendancetime,
            panel = function(x,...) {
                panel.histogram(x=x,...)
                m<-round(mean(x,na.rm = TRUE))
                panel.abline(v=m,col="red")
            }
)
```

Why not also add a line for where the 6 minute target is?

```
> histogram(firstpumparriving_attendancetime,
            panel = function(x,...) {
                panel.histogram(x=x,...)
                m<-round(mean(x,na.rm = TRUE))
                panel.abline(v=m,col="red")
                panel.abline(v=360,col="black")
            }
)
```

Next we can add some text to show the mean and difference in seconds:

```
> histogram(firstpumparriving_attendancetime,
            panel = function(x,...) {
                panel.histogram(x=x,...)
                m<-round(mean(x,na.rm = TRUE))
                d<-m-360
                panel.abline(v=m,col="red")
                panel.abline(v=360,col="black")
                panel.text(x=900,y=20,seconds_to_period(m))
                panel.text(x=900,y=17,d)
            }
)
```

To finish - if you have a big enough screen - we can plot the histogram for every borough simultaneously.

```
> histogram(~ firstpumparriving_attendancetime | incgeo_boroughname,
            scales=list(y=list(relation="free"), x=list(relation="free")),
            panel = function(x,...) {
                panel.histogram(x=x,...)
                m<-round(mean(x,na.rm = TRUE))
                d<-m-360
                panel.abline(v=m,col="red")
                panel.abline(v=360,col="black")
                panel.text(x=900,y=20,cex=0.7,seconds_to_period(m))
                panel.text(x=900,y=17,cex=0.7,d)
            }
 )
```

Now it should be possible to compare the latest performance figures from all boroughs across london to that before the closures. You can use the ODI London Fire Station closure analysis tool to find the historical data. Additionally, you can also see if the tool was able to accurately predict the results of the closures.

Learn more about using R

DataCamp has many more lessons on R, why not enrol for some of its courses?