# Practical 4: Linking R to the web (Web Mapping and Analysis)

*Hai Nguyen*

*18 November 2015*

# Practical: Linking R to the Web (Data)

## Reading data from the web

There are 2 ways that R can read data from the web: reading direct files or reading files served by an API. This section shows how to use R to read data using both ways.

### Reading direct files (csv/txt/json)

Datasets of school and colleges performances are available at (http://www.education.gov.uk/schools/performance/download_data.html)http://www.education.gov.uk/schools/performance/download_data.html (http://www.education.gov.uk/schools/performance/download_data.html) in PDF, CSV and Excel formats. For simplicity, we will try to read CSV format only, but XSL and PDF can also be downloaded and processed using R. To read the data as CSV file in R, we will need to use `read.csv(fileurl,options)` as follows:

where `fileurl` is the file location that can be in your file system (`C:/file.csv`) or on the web as in the code listing and `options` is a list of options specifying how the file is formatted. For example, in the above code listing, we specify that the columns are separated by comma as some CSV files might have columns separated by other characters. For a full description, type `?read.csv` in your R console.

The dataset will be imported as a `data.frame`. We can later display or process the dataset (or part of it). For example, we can list all schools' names where School Name contains `'Anfield'`

```
subset(liverpool_ks2_data,grepl('Anfield',SCHNAME))$SCHNAME
```

The result will be:

```
## [1] Anfield Road Primary School
## [2] Pinehurst Primary School Anfield
## [3] St Margaret's Anfield Church of England Primary School
## 123 Levels: Abbot's Lea School ... Woolton Primary School
```

### Reading files from a Web API/Servie (csv/txt/json)

Reading special file formats such as JSON (http://www.json.org) will require some external libraries such as `jsonlite`. In this section, we will use this library to retrieve a JSON file from a Web API (Application Programming Interface).

Generally a Web API is a service that receives requests or queries from users and returns a result via a web protocol (mainly HTTP). In this way, users can ask for and use data even without knowing how data are stored and processed. Due to the popularity of JavaScript in the WWW, JSON has become the most popular file format served by Web APIs.

In this section we will try to connect to the police.uk (https://data.police.uk/data)'s Data API. We will use `jsonlite` to download and parse the data (as JSON format). Firstly we will try to read all street-level crimes around the University during 07/2015 (API docs for this data are available at (https://data.police.uk/docs/method/crime-street)https://data.police.uk/docs/method/crime-street (https://data.police.uk/docs/method/crime-street)).

```
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'
##
## The following object is masked from 'package:utils':
##
##     View
```

`jsonlite` will automatically convert the JSON object to a data frame. We can use this data frame for further analysis. For example, we can see how many crime cases within each crime category:

```
summary(factor(Liv_crime_0715$category))
```

```
##  anti-social-behaviour          bicycle-theft               burglary 
##                    445                        2                      35 
##  criminal-damage-arson                  drugs           other-crime 
##                     34                       27                       2 
##           other-theft          public-order                robbery 
##                     14                       12                       2 
##           shoplifting theft-from-the-person         vehicle-crime 
##                     10                        3                      15 
##         violent-crime 
##                     34
```

# Reading the whole web page (using RCurl)

So far we have been looking only at some popular file formats such as CSV and JSON. How about webpages? In theory, a webpage is just a HTML file (with some embedded or linked CSS and JavaScript). Fortunately, there is a library in R that allows us to download/upload any web resource/file that has a location (URL), `RCurl`. This library is an R-wrapper for the famous curl (http://curl.haxx.se) commandline tool.

We can read a website, for example a Wikipedia website easily:

```
library(RCurl)
```

```
## Loading required package: bitops
```

```
getURL('https://en.wikipedia.org/wiki/Visa_requirements_for_British_citizens')
```

However, as HTML is just a markup language (i.e., a HTML document is made specifically for human to read, not machines), the data we received using `RCurl`. is a mixed of raw data and markups (for presenational purpose). To extract just data (or parts of data) without the markups, we can use another R package, namely `XML`. This package can parse an XML document (a HTML document is also an XML document) and return the data we wish to extract. For example, we can extract the table from the previous Wikipedia link (https://en.wikipedia.org/wiki/Visa_requirements_for_British_citizens)https://en.wikipedia.org/wiki/Visa_requirements_for_British_citizens (https://en.wikipedia.org/wiki/Visa_requirements_for_British_citizens) by using function `readHTMLTable` from `XML` package.

```
library(XML)
content <- getURL('https://en.wikipedia.org/wiki/Visa_requirements_for_British_citizens')
tables <- readHTMLTable(content)
```

Here the function `readHTMLTable` returns a list of tables. However, we are only interested in the first list (list of all countries). Let's have a look at this list:

```
first_table<- tables[[1]]
```

```
##                Country     Visa requirement
## 1          Afghanistan     Visa required[3]
## 2              Albania Visa not required[4]
## 3              Algeria     Visa required[5]
## 4              Andorra Visa not required[6]
## 5               Angola     Visa required[7]
## 6  Antigua and Barbuda Visa not required[8]
##   Notes (excluding departure fees)
## 1                                 
## 2                          90 days
## 3                                 
## 4                                 
## 5                                 
## 6
```

This dataframe has 3 columns, `Country`, `Visa requirement`, and `Notes`. Let's do a small exercise to see how many countries a British citizen can enter without a visa. We will need to look at the `` `Visa Requirement` `` column.

```
first_table$`Visa requirement`
```

```
##   [1] Visa required[3]
##   [2] Visa not required[4]
##   [3] Visa required[5]
##   [4] Visa not required[6]
##   [5] Visa required[7]
##   [6] Visa not required[8]
##   [7] Visa not required[9]
##   [8] Visa not required[11]
##   [9] eVisitor[14]
##  [10] EU !Visa not required[16]
##  [11] Visa required[17]
##  [12] Visa not required[22]
##  [13] Visa on arrival[25]
##  [14] Visa on arrival[28]
##  [15] Visa not required[32]
##  [16] Visa required[35]
##  [17] EU !Visa not required[37]
##  [18] Visa not required[38]
##  [19] Visa required[39]
##  [20] Visa required[40]
##  [21] Visa not required[43]
##  [22] Visa not required[45]
##  [23] Visa not required[46]
##  [24] Visa not required[47]
##  [25] Visa not required[49]
##  [26] EU !Visa not required[51]
##  [27] Visa on arrival[52]
##  [28] eVisa[53]
##  [29] Visa on arrival[56]
##  [30] Visa on arrival[57]
##  [31] Visa required[60]
##  [32] Visa not required[61]
##  [33] Visa on arrival[63]
##  [34] Visa required[64]
##  [35] Visa required[65]
##  [36] Visa not required[66]
##  [37] Visa required[67]
##  [38] Visa not required[68]
##  [39] Visa on arrival[70]
##  [40] Visa required[71]
##  [41] Visa required[72]
##  [42] Visa not required[73]
##  [43] EU !Visa not required[74]
##  [44] Visa required !Tourist Card required[75]
##  [45] EU !Visa not required[76]
##  [46] EU !Visa not required[77]
##  [47] EU !Visa not required[78]
##  [48] Visa on arrival[79]
##  [49] Visa not required[80]
##  [50] Visa on arrival !Tourist Card on arrival[81]
##  [51] Visa not required[82]
##  [52] Visa on arrival[83]
##  [53] Visa not required[84]
##  [54] Visa required[85]
##  [55] Visa required[86]
##  [56] EU !Visa not required[87]
##  [57] Visa on arrival[88]
##  [58] Visa not required[90]
##  [59] EU !Visa not required[91]
##  [60] EU !Visa not required[92]
##  [61] e-Visa[93]
```

```
##  [62] Visa not required[94]
##  [63] Visa not required[95]
##  [64] EU !Visa not required[97]
##  [65] Visa required[98]
##  [66] EU !Visa not required[99]
##  [67] Visa not required[100]
##  [68] Visa not required[101]
##  [69] Visa required[102]
##  [70] Visa on arrival[103]
##  [71] Visa not required[104]
##  [72] Visa not required[105]
##  [73] Visa not required[106]
##  [74] EU !Visa not required[107]
##  [75] EU !Visa not required[108]
##  [76] e-Tourist Visa[109]
##  [77] Visa not required[111]
##  [78] Visa required[113]
##  [79] Visa required[114]
##  [80] EU !Visa not required[115]
##  [81] Visa not required[116]
##  [82] EU !Visa not required[117]
##  [83] eVisa[118]
##  [84] Visa not required[119]
##  [85] Visa not required[120]
##  [86] Visa on arrival !Free visa on arrival[121][122]
##  [87] Visa not required[124]
##  [88] eVisa[126]
##  [89] Visa not required[127]
##  [90] Visa required[128]
##  [91] Visa not required[129]
##  [92] Visa on arrival !Free visa on arrival[130]
##  [93] Visa not required[131]
##  [94] Visa on arrival[134]
##  [95] EU !Visa not required[135]
##  [96] Visa on arrival[136]
##  [97] Visa not required[137]
##  [98] Visa required[138]
##  [99] Visa required[139]
## [100] EU !Visa not required[140]
## [101] EU !Visa not required[141]
## [102] EU !Visa not required[142]
## [103] Visa not required[143]
## [104] Visa on arrival !Free visa on arrival[144]
## [105] Visa on arrival[145]
## [106] Visa not required[146]
## [107] Visa on arrival !Free visa on arrival[147]
## [108] Visa required[148]
## [109] EU !Visa not required[149]
## [110] Visa on arrival !Free visa on arrival[150]
## [111] Visa on arrival[151]
## [112] Visa not required[152]
## [113] Visa not required[153]
## [114] Visa not required[155]
## [115] Visa not required[156]
## [116] Visa not required[157]
## [117] Visa required[158]
## [118] Visa not required[159]
## [119] Visa not required[160]
## [120] Visa required[161]
## [121] Visa not required[165]
## [122] Visa required[168]
## [123] Visa on arrival[169]
## [124] EU !Visa not required[171]
```

```
## [125] Visa not required[172]
## [126] Visa not required[173]
## [127] Visa required[174]
## [128] Visa required[175]
## [129] EU !Visa not required[176]
## [130] Visa on arrival[177]
## [131] Visa required[178]
## [132] Visa on arrival !Free visa on arrival[181]
## [133] Visa not required[182]
## [134] Visa on arrival !Free visa on arrival[183][184]
## [135] Visa not required[185]
## [136] Visa not required[186]
## [137] Visa not required[187]
## [138] EU !Visa not required[188]
## [139] EU !Visa not required[189]
## [140] Visa on arrival[190]
## [141] EU !Visa not required[191]
## [142] Visa required[192]
## [143] Visa on arrival[193]
## [144] Visa not required[194]
## [145] Visa not required[195]
## [146] Visa not required[196]
## [147] Visa on arrival !Free Entry Permit on arrival[197]
## [148] Visa not required[198]
## [149] Visa not required[199]
## [150] Visa required[200]
## [151] Visa not required[201]
## [152] Visa not required[202]
## [153] Visa on arrival !Free Visitor's Permit on arrival[203]
## [154] Visa required[204]
## [155] Visa not required[205]
## [156] EU !Visa not required[206]
## [157] EU !Visa not required[207]
## [158] Visa on arrival !Free Visitor's permit on arrival[208]
## [159] Visa on arrival[209]
## [160] Visa not required[210]
## [161] Visa required[212]
## [162] EU !Visa not required[213]
## [163] Electronic Travel Authorization[214]
## [164] Visa required[216]
## [165] Visa on arrival !Tourist Card on arrival[217]
## [166] Visa not required[218]
## [167] EU !Visa not required[219]
## [168] EU !Visa not required[220]
## [169] Visa required[221]
## [170] Visa on arrival[222]
## [171] Visa on arrival[223]
## [172] Visa not required[224]
## [173] Visa on arrival[227]
## [174] Visa on arrival[229]
## [175] Visa on arrival !Free visa on arrival[230]
## [176] Visa not required[231]
## [177] Visa not required[232]
## [178] eVisa[233]
## [179] Visa required[234]
## [180] Visa on arrival !Free visa on arrival[235]
## [181] Visa on arrival[236]
## [182] Visa not required[237]
## [183] Visa on arrival !Free visa on arrival[241]
## [184] Visa not required !Visa Waiver Program[242]
## [185] Visa not required[244]
## [186] Visa required[245]
## [187] Visa not required[247]
```

```
## [188] Visa not required[248]
## [189] Visa not required[249]
## [190] Visa not required[251]
## [191] Visa required[253]
## [192] Visa on arrival[254]
## [193] Visa on arrival[256]
## 193 Levels: e-Tourist Visa[109] ... Visa required[98]
```

There can be multiple ways to achieve this task, but to keep it simple, we will search for keyword *"not required"* using `grepl` .

```
not_required <- subset(first_table,grepl('not required',first_table$`Visa requirement`))
nrow(not_required)
```

```
## [1] 104
```

In total, there are 104 countries where a British citizen can go without a visa. Note that this list is not 100% correct as we ignored the e-visa, but it can give you a taste of mining open data sources available on the web in a semi-structured formats such as HTML tables.

# API example - Twitter

## Functions

In any programming languages, functions are always important. The most popular case you might need functions is when you want to re-use a block of code. Another case is to use function as an interface to exchange data from programs to programs or within a program. Some languages, such as R, allow you to use a function as a parameter of another function (this type of functions is referred to as higher-order functions).

A function in R has the following format:

```
function_name <- function(arg1, arg2, ... ){
  # doing something
return(result)
}
```

where `function_name` represents the unique name of your function, `argi` is the argument *i* (a function can take none or multiple arguments), and `return` is an optional statement within the function that returns the result.

If there is no `return` statement the function will return the last expression. Now let's create a function with different return statements in R:

```
add_function.1 <- function(a,b){return(a+b)}
add_function.2 <- function(a,b){a+b}
add_function.3 <- function(a,b){r <- a+b; r}
add_function.4 <- function(a,b){r <- a+b}
add_function.1(1,2)
```

```
## [1] 3
```

```
add_function.2(1,2)
```

```
## [1] 3
```

```
add_function.3(1,2)
```

```
## [1] 3
```

```
add_function.4(1,2)
```

We can see that all versions return intended correct results, except `add_function.4` which does not have an expression (note that `r <- a+b` is a statement, not an expression, in the sense that it does not return a value).

A function can be named or not (when there is no name, it is referred to as an anonymous function). Anonymous functions are usually used as an argument of another function (a.k.a. higher-order function). For example, we can create an anonymous function as follows:

```
list <- c(1:5)
result <- sapply(list,function(x){ return(x*2)})
result
```

```
## [1]  2  4  6  8 10
```

In this example, function `sapply` is a higher-order function and the function that doubles the argument is the anonymous function. Other, but not all, higher-order functions in R are `apply, sapply, mapply, Filter`…

## The TwitteR package

In this section we will learn how to access the *Twitter API* in R using the `twitteR` package. Firstly, you need to create a Twitter app and obtain neccessary credentials. Go to (https://dev.twitter.com/oauth/overview/application-owner-access-tokens)https://dev.twitter.com/oauth/overview/application-owner-access-tokens (https://dev.twitter.com/oauth/overview/application-owner-access-tokens) and get the access and consumer token/secret (you will need a Twitter Account to do this).

After having all the app's details, you can install `twitteR` and set up a Twitter session as follows.

```
install.packages("twitteR")
library("twitteR")
# Go to https://dev.twitter.com/oauth/overview/application-owner-access-tokens and get the access and consum
consumer_key <- 'your consumer key'
consumer_secret <- 'your consumer secret'
access_token <- 'your access token'
access_secret <- 'your access secret'
```

```
## [1] "Using direct authentication"
```

There are many things you can do with the Twitter API (full details available at: (https://dev.twitter.com/rest/public)https://dev.twitter.com/rest/public (https://dev.twitter.com/rest/public)). For example, you can access details of a Twitter user such as name, followers, friends, status, etc. For more details, have a look at the package documentation: (https://cran.r-project.org/web/packages/twitteR/twitteR.pdf)https://cran.r-project.org/web/packages/twitteR/twitteR.pdf (https://cran.r-project.org/web/packages/twitteR/twitteR.pdf).

Let's try to get some details of the university Twitter's account.

```
# firstly we need to search for the user
LiverpoolUni <- getUser('LivUni')
# example of getting attributes from a user object
LiverpoolUni$name
```

```
## [1] "Uni of Liverpool"
```

```
LiverpoolUni$friendsCount
```

```
## [1] 872
```

```
LiverpoolUni$description
```

```
LiverpoolUni$lastStatus
```

```
# example of calling a methods from a user object -- get 50 followers
followers <- LiverpoolUni$getFollowers(n=50)
head(followers)
```

```
## $`756663138`
## [1] "Goodkindles"
##
## $`50622068`
## [1] "weareweb"
##
## $`1617722742`
## [1] "fauxcrumpet"
##
## $`3708888796`
## [1] "SkeneHire"
##
## $`464258998`
## [1] "lostonbecca"
##
## $`4417743743`
## [1] "PropertiesUSUK"
```

Next step is to use `lapply` and create a function to retrieve a list of locations of the followers.

```
follower_locations <- lapply(followers,function(u) u$location)
```

Then filter out the empty locations (number of character is 0):

```
follower_locations <- Filter(function(l) nchar(l)>0, follower_locations)
head(follower_locations)
```

```
## $`756663138`
## [1] "USA"
##
## $`50622068`
## [1] "North West, England"
##
## $`1617722742`
## [1] "UK"
##
## $`3708888796`
## [1] "Blackburn, England"
##
## $`464258998`
## [1] "probably sleeping"
##
## $`3389950709`
## [1] "London, England  - SA"
```

Now we have a list of locations as text. However, to do any mapping, we will need these locations geocoded as either latitude/longitude or easting/northing. As the locations can be from around the world, we will geocode them into long/lat. Fortunately, R has the `ggmap` providing us the `geocode` function. Again, you will need to install this library using `install.packages("ggmap")`:

```
# get geocoding for locations
library("ggmap")
```

```
## Warning: package 'ggmap' was built under R version 3.2.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.2.3
```

```
follower_lnglat <- lapply(follower_locations,function(l) geocode(l))
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=USA&sensor=false
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=North%20West,%20England&s
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=UK&sensor=false
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Blackburn,%20England&sens
```

```
## Warning: geocode failed with status ZERO_RESULTS, location = "probably
## sleeping"
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=London,%20England%20%20-%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Liverpool&sensor=false
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=%E5%90%8D%E5%8F%A4%E5%B1%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=South%20Sefton%20College%
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Manchester,%20Birmingham,
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=UNIGE%20Geneva%20&%20UNI
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=London&sensor=false
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Liverpool,%20England&sen
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Liverpool,%20England&sen
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ampthill,%20England&sens
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Manchester/liverpool%20&
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=MediaCityUK&sensor=false
```

```
## Warning: geocode failed with status ZERO_RESULTS, location = "Piirus blog"
```

```
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=United%20Kingdom&sensor=
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=St%20Neots,%20England%20
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Manchester&sensor=false
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Winchester,%20Hampshire,
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=United%20Kingdom&sensor=
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=England,%20United%20Kingd
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Thame,%20England&sensor=
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=London,%20England&sensor
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Liverpool%20City%20regio
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Liverpool&sensor=false
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Stratford-Upon-Avon,%20E
## .Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=edinburgh%20&sensor=fals
```

Finally you will create a data frame, namely `geocoded` with the location names, latitudes and longitudes:

```
# convert the list of data frame into a data frame
geocoded <- do.call('rbind',follower_lnglat)
geocoded$place_name <- unlist(follower_locations)
```

We can export this data frame as a csv file (similar to the way we read a csv file in the first section of this practical):

```
#write result to a csv file for visualisation. note that the file will be saved in your working directory
# to check your wd, run getwd()
write.csv(geocoded,file="LivUniFollowers.csv")
```

Finally, to visualise your result, you can use an online mapping service at (http://www.mapsdata.co.uk/mapsdataapp)http://www.mapsdata.co.uk/mapsdataapp (http://www.mapsdata.co.uk/mapsdataapp) to import your csv file and visual the map (with clusters, bubble map, heatmap, etc.). In the next practical, you will learn how to create these maps by yourself using the *Leaflet* library.

# Reading and Resources:

Kitchin, R. (2014) The Data Revolution: Big Data, Open Data, Data Infrastructures & Their Consequences. London: Sage. Goodchild, M.F. (2007) Citizens as sensors: the world of volunteered geography. GeoJournal 69 (4): 211–221. Hacklay, M, Weber, P. (2008) OpenStreetMap: User-Generated Street Maps Pervasive Computing, IEEE (Volume:7, Issue: 4) Available from (http://discovery.ucl.ac.uk/13849/1/13849.pdf)http://discovery.ucl.ac.uk/13849/1/13849.pdf (http://discovery.ucl.ac.uk/13849/1/13849.pdf) Russell, M.A. (2013) Mining the Social Web. Second Edition. Sebastopol, CA: O'Reilly Media.