MSc Web and Social Media Analytics

Tutorial Pack 2

(To be completed during LW2 tutorial)

(90 minutes)

LEARNING OBJECTIVES

- **Familiarise students with key web concepts.**
- **To provide practical experience of using R to perform basic analysis of web traffic data.**

LEARNING OUTCOMES

- **By the end of this tutorial students will have.**
  - **Increased their understanding of the application of GET parameters.**
  - **Familiarised themselves with important R concepts and functions.**
  - **Developed experience importing and loading R packages.**
  - **Increased their awareness of the charting functionality available in R.**
  - **Enhanced their understanding of R functions.**

RESOURCES AND TOOLS REQUIRED

- **weblog.csv**
- **R (v4) or RStudio via appsanywhere**

IMPORTANT:

This pack is designed for you to go at your own speed and gets progressively more difficult. At the end of each section there is a series of practice questions and exercises – you should attempt to answer **all questions**.

Any questions you do not complete today should be completed before your next tutorial.

Study Notes

In the LW1 lecture we reviewed the technical operation of the web and the role of web servers in providing a source of data we can use for web analytics.

All resources on the web (e.g., pages, images, or videos) are accessed using their URI (Uniform Resource Identifier). Most of the time we do not enter the URI directly, rather we use a search engine or click on a link to obtain the appropriate URI. You should note that, whilst URIs are themselves unique, multiple different URIs can be configured to point to the same resource on the web.

Uniform Resource Identifier

Every resource available on the web can be accessed through its URI. An example of a URI is "http://www.amazon.co.uk/departments/sports.html?id=2912"

The URI is made up of several components.

The scheme: indicates the protocol that should be used to access the resource.
***The sub-domain**: a prefix of the domain name used to manage large sites with complex hierarchies.*
***amazon**:* the domain name, *identifies a specific organisation or entity.*
***co.uk**: the top-level domain, shows the country where the domain name is registered.*
path: the exact path to the resource on the web server
query: sets of parameters passed to this resource when it is retrieved

Suppose the University wanted to place a link on one of its web pages to another page called "About us". An appropriate URI used to create this link might resemble the following.

```
http://www.westminster.ac.uk/aboutus
```

If, however, we wanted to show a specific version of the about page when this link was clicked (for example in a different language), we could modify the query component of the URI so that one or more parameters are set. These parameters will be read by the webserver when deciding what content to display and how it should be formatted. For example, suppose we want anyone who clicks on this link to see a Chinese language version of the page we might choose to use the following URI.

```
http://www.westminster.ac.uk/aboutus?lang=cn
```

In this example we have only set one parameter, a variable called "lang" which is set to "cn". If we want to set more than one parameter, they need to be separated by the ampersand symbol (&), as shown below. In the context of web, these parameters are known as GET parameters.

```
http://www.westminster.ac.uk/aboutus?lang=cn&colourscheme=red&font=large
```

You should note that, by themselves, GET parameters will have no effect when used unless the website is itself configured to respond to them. You can try this yourself by visiting a site and adding your own custom GET parameters to the URL. GET parameters are however automatically logged by the web server whenever they are present.

Questions

Using your knowledge of URIs and GET parameters, attempt to answer the following questions.

Qnum	Question	ANSWER
1	Consider the following URI. https://www.abcd.ie/search.html Which part of this URI corresponds to the top-level domain component? Which country do you think this URI might originate?	
2	Consider the following URI https://www.amazon.co.uk/ref=nav_logo Which part of this URI corresponds to the path component?	
3	Suppose there are two websites that both offer the same product at the same price. The URI of store 1 is http://www.superstore.com whereas the URI of store 2 https://www.mystore.com Based on the URI alone, which store might you choose to buy the product assuming you will need to enter your credit card details?	
4	A company has recently launched a new version of its entire website, incorporating feedback from lots of different users. The new website contains the same content as before except that the format and colour scheme has changed. For the time being, the website would like to have both versions available as some users may want to use the old version or have issues with the new design. How might this be achieved without having to register a new domain name?	
5	What are the names of the variables set to 2 when a user clicks on the following URI? https://www.xyz.com?d=3&j=2&x=two&p=2	

One practical use of GET parameters involves the collection of clickstream data. This is a topic we will explore in more detail in the LW2 lecture. They are also used in **referral** and **affiliate marketing schemes**, whereby a business will pay a commission to another entity based on the number of people who they send to a website that go on to make a purchase.

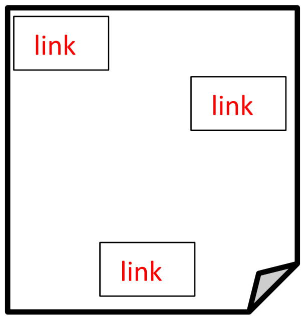
Affiliate Marketing using GET parameters

Online businesses regularly promote their sites by incentivising online celebrities and personalities to promote their products and services through their own websites. Each time a new customer places an order the online business will pay a commission, for example 5% of the value of the goods sold, to the celebrity or online personality that referred a customer to their site.

To be able to identify which celebrity of online personality referred a given customer, the business will create a URI that contains a set of parameters unique to each celebrity. The business will ask the celebrity or personality to use this specific URI in their marketing materials. When an order is placed, the business can check the GET parameters set to identify the celebrity or personality that is awarded the commission. For example, http://www.mystore.com?referrer=10291

Questions

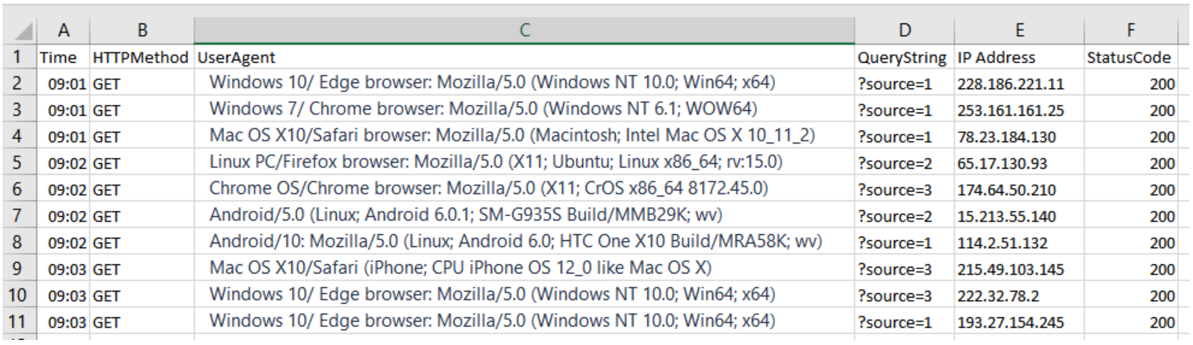
Using your knowledge of URIs and GET parameters, attempt to answer the following questions.

Qnum	Question	ANSWER
6	An online sports events company sells tickets to upcoming cricket matches. It largely advertises tickets using billboards placed at different locations. This includes on buildings and by the roadside. Explain one method it could use to gauge the number of people who saw an advertisement on a given billboard.	
7	Consider the following page.  The red link text indicates the placement of a clickable text link to a page called “Latest Offers”. The URI for all these links is currently. http://www.abc.com/latestoffers.html	

	Explain how these links could be modified understand where on the page a user clicked to access the latest offers. Illustrate your answer with an example.	

Web logging was one of the approaches that I suggested in LW1 could be used to collect data on user activity. The screenshot below shows a small extract from a web log containing ten entries across six fields. Many of these fields you might remember.

This data is available in a CSV (comma separated values) file named weblog.csv. This file can be found in the Week 2 folder on Blackboard. Each row represents a unique visitor.



	A	B	C	D	E	F
1	Time	HTTPMethod	UserAgent	QueryString	IP Address	StatusCode
2	09:01	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)	?source=1	228.186.221.11	200
3	09:01	GET	Windows 7/ Chrome browser: Mozilla/5.0 (Windows NT 6.1; WOW64)	?source=1	253.161.161.25	200
4	09:01	GET	Mac OS X10/Safari browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2)	?source=1	78.23.184.130	200
5	09:02	GET	Linux PC/Firefox browser: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0)	?source=2	65.17.130.93	200
6	09:02	GET	Chrome OS/Chrome browser: Mozilla/5.0 (X11; CrOS x86_64 8172.45.0)	?source=3	174.64.50.210	200
7	09:02	GET	Android/5.0 (Linux; Android 6.0.1; SM-G935S Build/MMB29K; wv)	?source=2	15.213.55.140	200
8	09:02	GET	Android/10: Mozilla/5.0 (Linux; Android 6.0; HTC One X10 Build/MRA58K; wv)	?source=1	114.2.51.132	200
9	09:03	GET	Mac OS X10/Safari (iPhone; CPU iPhone OS 12_0 like Mac OS X)	?source=3	215.49.103.145	200
10	09:03	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)	?source=3	222.32.78.2	200
11	09:03	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)	?source=1	193.27.154.245	200

Web logs in Practice

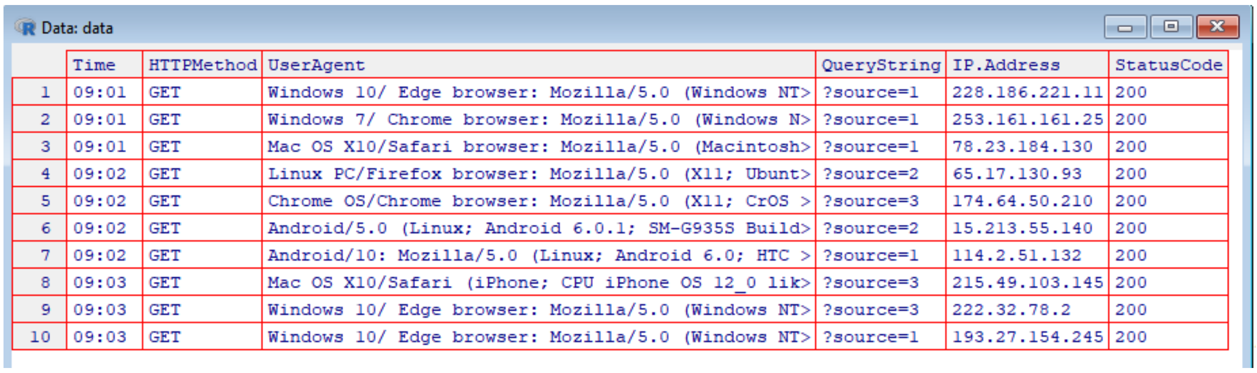
In real-life, the web log data would be significantly larger than in the example above. Potentially spanning many GiB's of data. For the purposes of this exercise, it is sufficient to walk through the theory and demonstrate the key techniques.

The first step in our analysis is to read in the CSV file into a new variable, in this case the variable is named "data". Recall from last week, you will need to update the path specified to point to the location where you downloaded this file.

```
data <- read.csv("C:\\Users\\Philip Worrall\\Desktop\\weblog.csv")
```

After loading data into R it is good practice to check that the data you have loaded in is fact the data that you intended to analyse. For this purpose R provides a helper function called **View**(). This function provides a visual, and scrollable, representation of the variable passed and can be used as a quick visual check.

```
> View(data)
```


Data: data

	Time	HTTPMethod	UserAgent	QueryString	IP.Address	StatusCode
1	09:01	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT>	?source=1	228.186.221.11	200
2	09:01	GET	Windows 7/ Chrome browser: Mozilla/5.0 (Windows N>	?source=1	253.161.161.25	200
3	09:01	GET	Mac OS X10/Safari browser: Mozilla/5.0 (Macintosh>	?source=1	78.23.184.130	200
4	09:02	GET	Linux PC/Firefox browser: Mozilla/5.0 (X11; Ubunt>	?source=2	65.17.130.93	200
5	09:02	GET	Chrome OS/Chrome browser: Mozilla/5.0 (X11; CrOS >	?source=3	174.64.50.210	200
6	09:02	GET	Android/5.0 (Linux; Android 6.0.1; SM-G935S Build>	?source=2	15.213.55.140	200
7	09:02	GET	Android/10: Mozilla/5.0 (Linux; Android 6.0; HTC >	?source=1	114.2.51.132	200
8	09:03	GET	Mac OS X10/Safari (iPhone; CPU iPhone OS 12_0 lik>	?source=3	215.49.103.145	200
9	09:03	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT>	?source=3	222.32.78.2	200
10	09:03	GET	Windows 10/ Edge browser: Mozilla/5.0 (Windows NT>	?source=1	193.27.154.245	200

Similarly, the **head**() and **tail**() functions allow the programmer to view the first or last N rows respectively. R programmers will typically use these functions when working with large datasets or time ordered data, since they will want to confirm the period the dataset spans.

```
> head(data,2)
   Time                                                     UserAgent QueryString     IP.Address
1 09:01 Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=1 228.186.221.11
2 09:01     Windows 7/ Chrome browser: Mozilla/5.0 (Windows NT 6.1; WOW64)   ?source=1 253.161.161.25
```

```
> tail(data,2)
    Time                                                     UserAgent QueryString     IP.Address
9  09:03 Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=3    222.32.78.2
10 09:03 Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=1 193.27.154.245
```

When importing datasets it is also good practice to check for the presence of duplicate rows. Duplicate rows can arise for a variety of reasons, including mistakes in data entry or files being combined, but if not dealt with they can cause errors in later analysis.

One way to identify rows or data elements that are distinct is by using the **unique**() function. The unique function will return all distinct elements of a variable passed as a parameter, be it a vector or a data frame.

When the unique function is applied to a vector, we see only the distinct elements.

```
> unique(c("a","b","c","c"))
[1] "a" "b" "c"
```

When the unique function is applied a data frame, we see only the distinct rows.

```
> unique(data)
    Time                                                         UserAgent QueryString     IP.Address
1  09:01          Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=1 228.186.221.11
2  09:01              Windows 7/ Chrome browser: Mozilla/5.0 (Windows NT 6.1; WOW64)   ?source=1 253.161.161.25
3  09:01  Mac OS X10/Safari browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2)   ?source=1  78.23.184.130
4  09:02 Linux PC/Firefox browser: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0)   ?source=2    65.17.130.93
5  09:02          Chrome OS/Chrome browser: Mozilla/5.0 (X11; CrOS x86_64 8172.45.0)   ?source=3  174.64.50.210
6  09:02                 Android/5.0 (Linux; Android 6.0.1; SM-G935S Build/MMB29K; wv)   ?source=2  15.213.55.140
7  09:02  Android/10: Mozilla/5.0 (Linux; Android 6.0; HTC One X10 Build/MRA58K; wv)   ?source=1   114.2.51.132
8  09:03                   Mac OS X10/Safari (iPhone; CPU iPhone OS 12_0 like Mac OS X)   ?source=3 215.49.103.145
9  09:03          Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=3     222.32.78.2
10 09:03          Windows 10/ Edge browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64)   ?source=1 193.27.154.245
```

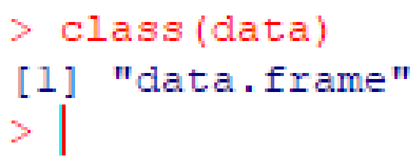
Questions

Using your knowledge of R and web logging, attempt to answer the following questions.

Qnum	Question	ANSWER
8	What key factors might affect the size of a website's web log file?	
9	In the web log file, what time zone is the "Time" field typically set to? Why is this important?	
10	Load the above data into R and use the view function to check that the correct data has been loaded.	
11	Without supplying the N parameter to the head() function, how many rows are returned?	
12	Using an appropriate function, show the last 4 rows	
13	Write an R code to determine if there are any duplicate rows in your dataset. Hint: you might find nrow() useful.	

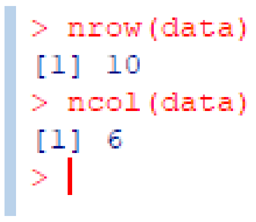
The **data frame** is an important concept in the R statistical package. Like matrices, which we saw last week, data frames resemble two dimensional arrays although they can contain mixtures of different data types. You can think of a data frame as analogous to an Excel spreadsheet. Data frames are often used in place of matrices because they enable the user to refer to specific columns and rows by their name.

We can confirm that the data we ready in using the **read_csv()** function is a data frame using the **class()** function from last week.



```
> class(data)
[1] "data.frame"
> 
```

Two useful functions to determine the number of columns and rows in a data frame include the **ncol()** and **nrow()** functions. Both functions except a data frame as an input variable.



```
> nrow(data)
[1] 10
> ncol(data)
[1] 6
> 
```

The **names()** function applied to a data frame returns a vector of column names

```
> names(data)
[1] "Time"        "HTTPMethod"  "UserAgent"   "QueryString" "IP.Address"
[6] "StatusCode"
> 
```

Internally, R stores a data frame a list of column vectors. Both of which we spoke about in LW1. Once we know the name of a column, we are interested in we can use the $ syntax to access individual column vectors.

```
> data$UserAgent
 [1] "Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
 [2] "Windows 7/ Chrome browser:\xa0Mozilla/5.0 (Windows NT 6.1; WOW64) "
 [3] "Mac OS X10/Safari browser:\xa0Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2)"
 [4] "Linux PC/Firefox browser:\xa0Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) "
 [5] "Chrome OS/Chrome browser:\xa0Mozilla/5.0 (X11; CrOS x86_64 8172.45.0) "
 [6] "Android/5.0 (Linux; Android 6.0.1; SM-G935S Build/MMB29K; wv) "
 [7] "Android/10: Mozilla/5.0 (Linux; Android 6.0; HTC One X10 Build/MRA58K; wv)"
 [8] "Mac OS X10/Safari (iPhone; CPU iPhone OS 12_0 like Mac OS X) "
 [9] "Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
[10] "Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
> 
```

Slicing is another technique you may come across when reading R code. Previously when we wanted to access an individual element within a vector, we used indexing notation [] to specify the position in the vector we wanted to read. In contrast, when working with a data frame we will typically be interested in a subset of rows **or** columns, or indeed rows **and** columns. In this case we specify two indexes to form a slice.

For example, to select only the first column across all rows.

```
> data[,1]
 [1] "09:01" "09:01" "09:01" "09:02" "09:02" "09:02" "09:02" "09:03" "09:03" "09:03"
> 
```

To select only row 2, across columns 1,2 and 3.

```
> data[2,1:3]
   Time HTTPMethod                                                          UserAgent
2 09:01        GET Windows 7/ Chrome browser:\xa0Mozilla/5.0 (Windows NT 6.1; WOW64)
> 
```

To select rows 2, 4 and 6 across columns 4 and 5.

```
> data[c(2,4,6),4:5]
  QueryString     IP.Address
2   ?source=1 253.161.161.25
4   ?source=2   65.17.130.93
6   ?source=2   15.213.55.140
> 
```

Zero variance

If you are familiar with machine learning or predictive modelling you may have come across the concept of zero variance before. It is standard practice to remove columns with only 1 distinct value as part of the initial data pre-processing and import process. The main reason for doing this is to make models simpler, use less data and filter out fields that do not help to increase the explanatory power of a predictive model.

Questions

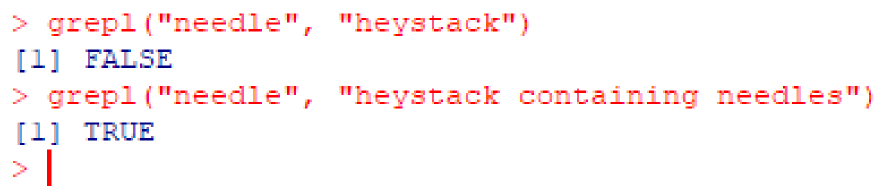
Using your knowledge of R and data frames, attempt to answer the following questions.

Qnum	Question	ANSWER
14	Determine if any fields in your dataset have zero variance, i.e., only hold a single value.	
15	Remove any identified columns with zero variance from your dataset.	

Suppose we are interested in the device used to access a site, perhaps we want to know how many users access the site from an android device. One possibility would be to use the user agent field to extract this information. However, each entry appears unique because the user agent field stores build and version numbers which makes aggregation problematic.

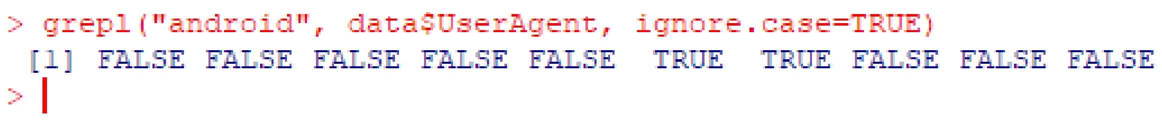
One strategy is to recode the user agent field such that android and non-android users and separated. We can test for the logical presence of the word android in a field using the **grepl()** function. The grepl() function accepts two parameters, the pattern to search for and the string to search. The result of grepl() is always either True or False.

For example.



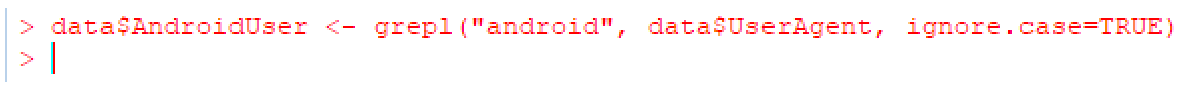
```
> grepl("needle", "heystack")
[1] FALSE
> grepl("needle", "heystack containing needles")
[1] TRUE
> 
```

Applying this to our dataset can be done as follows, notice how the third parameter "ignore.case=True" allows case insensitive matching. From the results we can see that we have just two android users out of a possible 10.



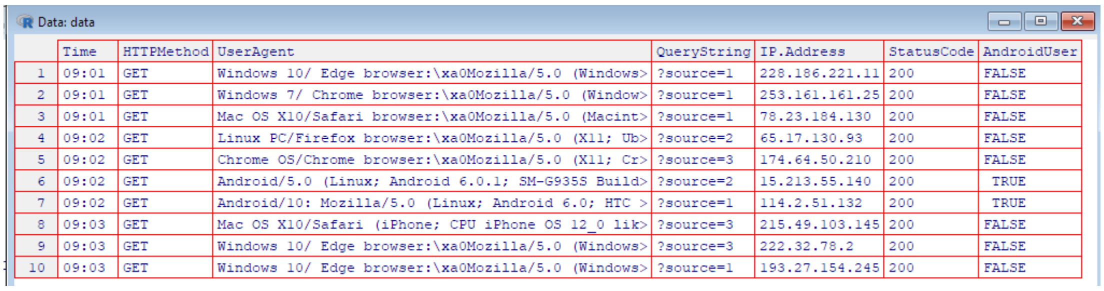
```
> grepl("android", data$UserAgent, ignore.case=TRUE)
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
> 
```

The results can be saved back to the original data frame by assigning them to a new column.



```
> data$AndroidUser <- grepl("android", data$UserAgent, ignore.case=TRUE)
> 
```

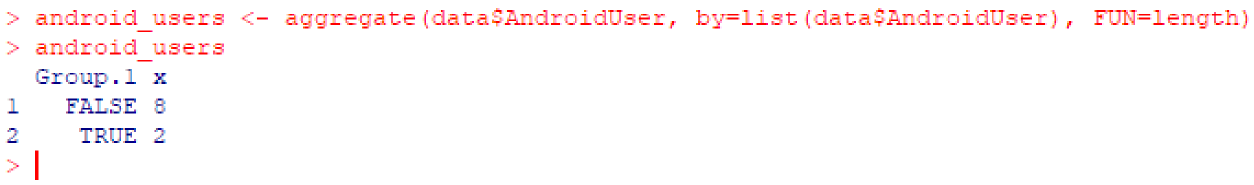
Using the View() function we can check that the new column has been added and populated.



Data: data

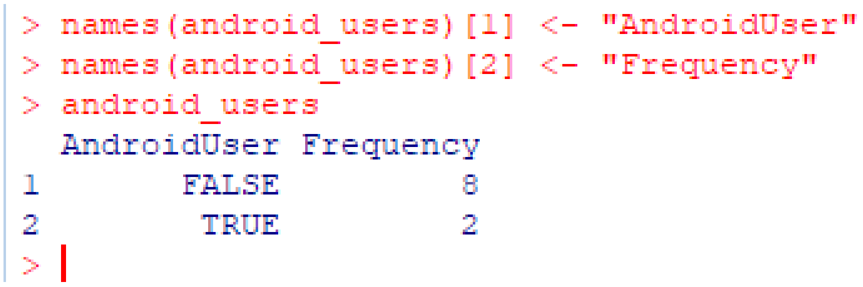
	Time	HTTPMethod	UserAgent	QueryString	IP.Address	StatusCode	AndroidUser
1	09:01	GET	Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows>	?source=1	228.186.221.11	200	FALSE
2	09:01	GET	Windows 7/ Chrome browser:\xa0Mozilla/5.0 (Window>	?source=1	253.161.161.25	200	FALSE
3	09:01	GET	Mac OS X10/Safari browser:\xa0Mozilla/5.0 (Macint>	?source=1	78.23.184.130	200	FALSE
4	09:02	GET	Linux PC/Firefox browser:\xa0Mozilla/5.0 (X11; Ub>	?source=2	65.17.130.93	200	FALSE
5	09:02	GET	Chrome OS/Chrome browser:\xa0Mozilla/5.0 (X11; Cr>	?source=3	174.64.50.210	200	FALSE
6	09:02	GET	Android/5.0 (Linux; Android 6.0.1; SM-G935S Build>	?source=2	15.213.55.140	200	TRUE
7	09:02	GET	Android/10: Mozilla/5.0 (Linux; Android 6.0; HTC >	?source=1	114.2.51.132	200	TRUE
8	09:03	GET	Mac OS X10/Safari (iPhone; CPU iPhone OS 12_0 lik>	?source=3	215.49.103.145	200	FALSE
9	09:03	GET	Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows>	?source=3	222.32.78.2	200	FALSE
10	09:03	GET	Windows 10/ Edge browser:\xa0Mozilla/5.0 (Windows>	?source=1	193.27.154.245	200	FALSE

In R the **aggregate**() function can be used to group data and apply a common function on each group. In the example below I am using the aggregate function to group all the rows by their corresponding user agent. The function that I would like to apply to each group is the **length()** function, which effectively returns the number of rows in each group. The output of the aggregate function is a data frame containing the grouped columns and the result of the function applied to each group.



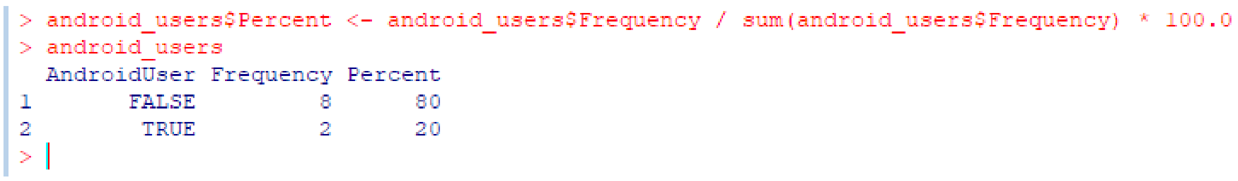
```
> android_users <- aggregate(data$AndroidUser, by=list(data$AndroidUser), FUN=length)
> android_users
  Group.1 x
1   FALSE 8
2    TRUE 2
> 
```

Unfortunately, the aggregate function doesn't return very useful column names as part of its output data frame. It is however easy to fix that using names() function. In this case I change the first column's name to "AndroidUser" and the second to "Frequency".



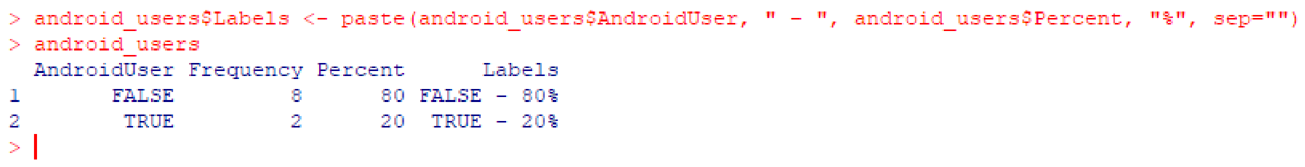
```
> names(android_users)[1] <- "AndroidUser"
> names(android_users)[2] <- "Frequency"
> android_users
  AndroidUser Frequency
1       FALSE         8
2        TRUE         2
> 
```

In the screenshot below I have added a new column to the data frame containing the grouped frequencies. The column is called "Percent" and it is calculated by taking the frequency column of each group and dividing it by the summation of all frequencies across all groups. In effect this calculates the percentage of visits according to whether an android device was used.



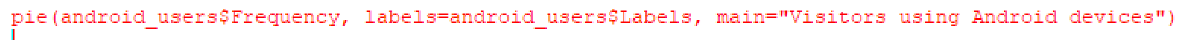
```
> android_users$Percent <- android_users$Frequency / sum(android_users$Frequency) * 100.0
> android_users
  AndroidUser Frequency Percent
1       FALSE         8      80
2        TRUE         2      20
> 
```

With the help of the **paste**() function, which joins individual character strings into one large string separated by a specified string separator, I have created a new derived column called "Labels". The label for each row is based on the AndroidUser field and the percentage of visits.

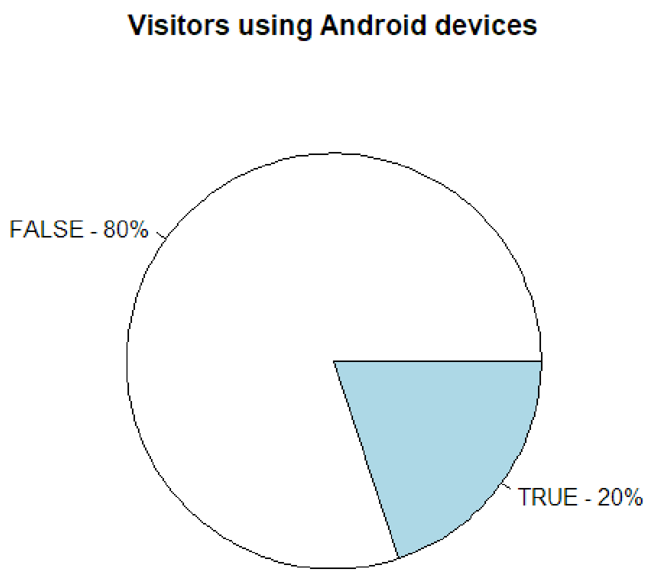


```
> android_users$Labels <- paste(android_users$AndroidUser, " - ", android_users$Percent, "%", sep="")
> android_users
  AndroidUser Frequency Percent      Labels
1       FALSE         8      80 FALSE - 80%
2        TRUE         2      20  TRUE - 20%
> 
```

Finally it remains to plot this data using **pie**() function. The pie function accepts the data to be plotted in the first parameter. The labels used for each segment of the pie come from the Labels column I created in the previous step and are specified using the Labels parameter. The main parameter is responsible for setting the title.

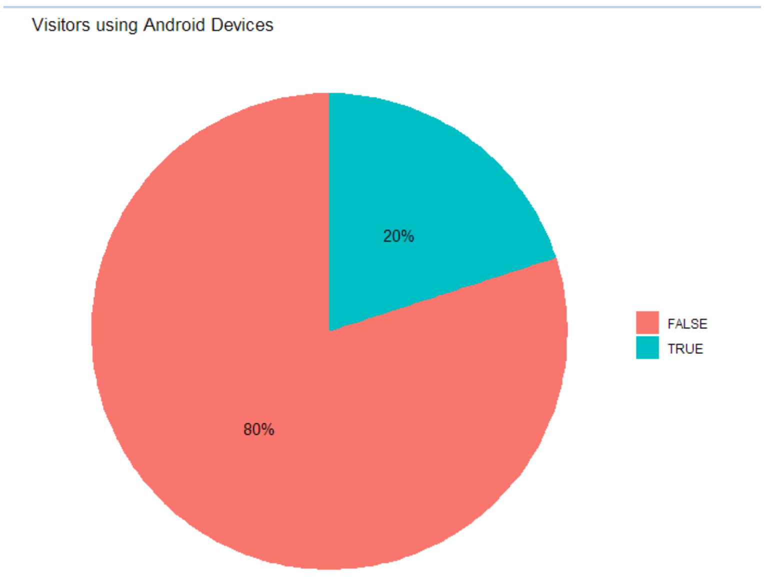


```
pie(android_users$Frequency, labels=android_users$Labels, main="Visitors using Android devices")
```



For more customisation, students on the data visualisation module may elect to plot the data using **ggplot().** In that case a possible way uses the following section of code:

```
ggplot(android_users, aes(x="", y=Frequency, fill=AndroidUser)) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +
  theme_void() +
  ggtitle("Visitors using Android Devices") +
  geom_text(aes(label = paste0(Percent, "%")), position = position_stack(vjust=0.5)) +
  labs(x = NULL, y = NULL, fill = NULL)
```



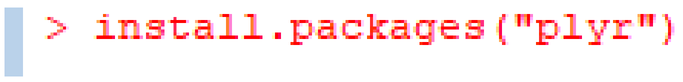
Questions

Using your knowledge of R and data frames, attempt to answer the following questions.

Qnum	Question	ANSWER
16	Follow the steps in the previous section to create a new column in your data frame that records whether a visit came from a user on a Windows device.	
17	Modify the code used previously to generate a pie chart showing the proportion of users that accessed the site using a Windows device.	

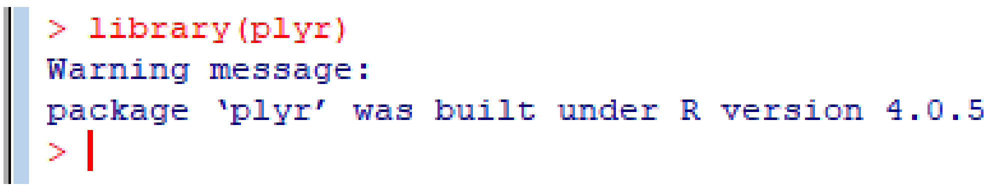
So far, we have used functions that are part of R's base installation. We can also use functions that have been developed externally by other users of R. Groups of existing functions are often organised into packages that can be downloaded and installed to our local R installation. To install an external package we can use the **install.packages**() function and pass the name of the package we would like to install as the first parameter.

The package I am going to use for the next step is called "plyr" . This package includes a range of helper functions commonly used for data science and analysis.



```
> install.packages("plyr")
```

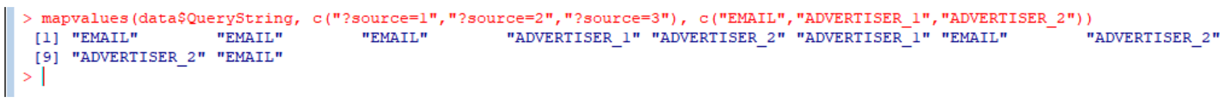
Before functions in a package can be used, the package needs to be loaded into R using the **library**() function.



```
> library(plyr)
Warning message:
package 'plyr' was built under R version 4.0.5
> 
```

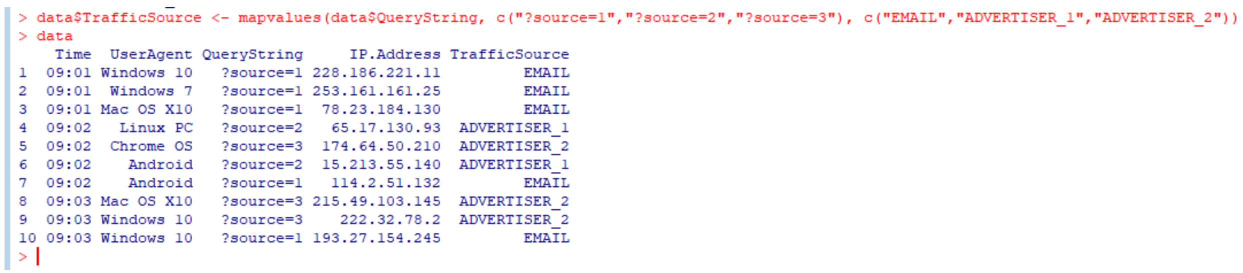
A useful function in the plyr package is called **mapvalues**(). It can be used as part of the data recoding process to convert a set of values into another. In this case our query string represents from which source the user clicked a link to access a web page.

The “?source=1” link was only used in promotional emails where as “?source=2” and “?source=3” was given to two separate advertisers to use. It is much clearer to the user of the analysis if I map these three abstract values into the names of the different traffic sources. An example of how to do this is shown below.



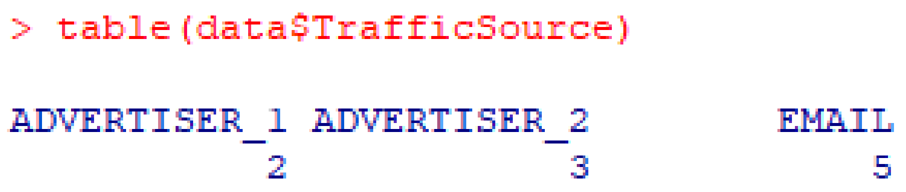
```
> mapvalues(data$QueryString, c("?source=1","?source=2","?source=3"), c("EMAIL","ADVERTISER_1","ADVERTISER_2"))
 [1] "EMAIL"        "EMAIL"        "EMAIL"        "ADVERTISER_1" "ADVERTISER_2" "ADVERTISER_1" "EMAIL"        "ADVERTISER_2"
 [9] "ADVERTISER_2" "EMAIL"
> 
```

With the mapvalues function applied I have created a new column called TrafficSource.



```
> data$TrafficSource <- mapvalues(data$QueryString, c("?source=1","?source=2","?source=3"), c("EMAIL","ADVERTISER_1","ADVERTISER_2"))
> data
    Time  UserAgent QueryString     IP.Address TrafficSource
1  09:01 Windows 10   ?source=1 228.186.221.11         EMAIL
2  09:01  Windows 7   ?source=1 253.161.161.25         EMAIL
3  09:01 Mac OS X10   ?source=1  78.23.184.130         EMAIL
4  09:02   Linux PC   ?source=2   65.17.130.93  ADVERTISER_1
5  09:02  Chrome OS   ?source=3  174.64.50.210  ADVERTISER_2
6  09:02    Android   ?source=2  15.213.55.140  ADVERTISER_1
7  09:02    Android   ?source=1    114.2.51.132        EMAIL
8  09:03 Mac OS X10   ?source=3 215.49.103.145  ADVERTISER_2
9  09:03 Windows 10   ?source=3    222.32.78.2  ADVERTISER_2
10 09:03 Windows 10   ?source=1 193.27.154.245         EMAIL
> 
```

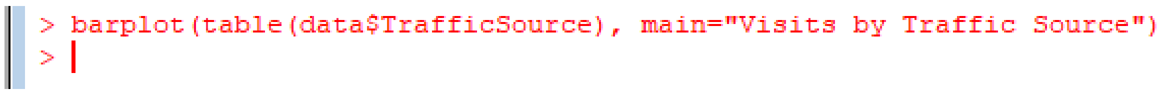
To present this data in a chart I could use the aggregate function, as used previously, but since we are only interested in the counts of one variable an alternative function is the **table**() function. The table function in R is designed to make it as easy as possible to quickly generate frequency tables.



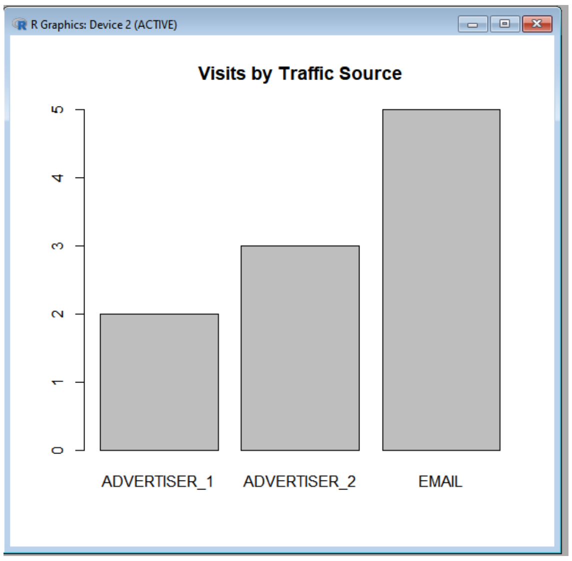
```
> table(data$TrafficSource)

ADVERTISER_1 ADVERTISER_2        EMAIL
           2            3            5
```

This time, instead of using a pie chart, I will use a bar chart to present this information. The first parameter to the **barplot**() function is the table containing the frequencies by category. The parameter “main” specifies the title of the chart.



```
> barplot(table(data$TrafficSource), main="Visits by Traffic Source")
> 
```



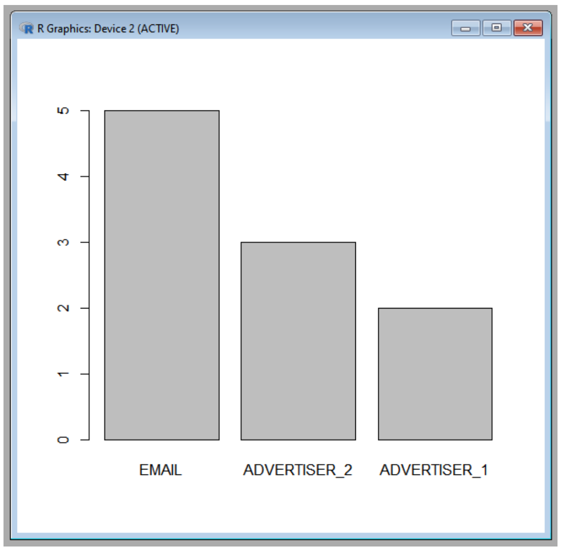
You will notice that the above bar chart displays the columns in the same order as the supplied frequency table. In this case the frequency table is organised alphabetically by column name. In many cases we may prefer to see the results in order of their frequencies. We can modify our code slightly and make use of the **order**() function to do this.

In the code below I have assigned my frequency table to the variable called “traffic_source_table”. This variable had its elements reorganised in decreasing frequency order by passing the variable “traffic_source_table” to the order function. You can see from the screenshots below that the EMAIL category now appears first since it has the highest frequency.

```
> traffic_source_table <- table(data$TrafficSource)
> traffic_source_table <- traffic_source_table[order(traffic_source_table, decreasing=TRUE)]
> traffic_source_table

       EMAIL ADVERTISER_2 ADVERTISER_1
           5            3            2
```

We can then replot the data to observe the new column ordering.



Questions

Using your knowledge of R, attempt to complete the following exercises.

Qnum	Question	ANSWER
18	Consider the following section of code. mapvalues(x, c("A","B",C"), c(1,5,9)) What will values of "B" in the variable named "x" get replaced with?	
19	Follow the steps in the previous section to create a bar chart of visits by traffic source.	
20	Look carefully at the following section of code. Can you explain what the order() function is doing and how the results of the order function are used? traffic_source_table[order(traffic_source_table, decreasing=True)]	
21	Modify your analysis to create a bar chart of visitors by SoftwareName, except this time use the table() function instead of the aggregate() function.	

If you have completed the exercises, there are some extension exercises in the Week 2 folder