Analysing large scale survey data using R

Pierre Walthéry, Nadia Kennar, Rihab Dahab

UK Data Service

Table of contents

Introduction

This guide provides an introduction to analysing large scale social survey dataset using R with examples from the British Social Attitudes Survey 2020. It is aimed at two categories of users:

1. Those outside higher education, or who do not have access to one commonly used commercial statistical software such as Stata, SPSS or SAS but who would like to conduct their own analysis beyond what is usually published by data producers such as the Office for National Statistics (for example statistics for specific groups of the population). This guide provides this group of users with a range of procedures that will help them produce straightforward and robust analyses tailored to their needs without spending unnecessary time on learning the inner workings of R.
2. More advanced users who are already familiar with other data analysis tools but who would like to learn how to carry out their analyses in R. The guide therefore focuses on providing succinct examples of common operations that most users carry out in the course of their research, including how to:

    - read in and open datasets.
    - do common data manipulation operations.
    - produce simple descriptive statistics or tabulations.
    - use survey weights.

1 What is R ?

R is a free, user developed, object-oriented statistical programming language that originates in the 'S' and 'S Plus' languages developed during the 1970s and 1980s. It has a large audience in the science and statistics communities and is increasingly used in the social sciences for teaching and research purposes.

Anyone can install and use R without charge, and to some extent contribute to and amend the existing program itself. R can be downloaded from the Comprehensive R Archive Network (CRAN) website. Installation instructions as well as guides, tutorials and FAQ are available on the CRAN website.

R is particularly favoured by users who want to develop their own statistical functions or implement technical advances that are not yet available in commercial packages. The existence of a vast number of user written packages (17,672 at the time of writing this guide) is one of the great strengths of R. Users who want to contribute should be aware that in order to be part of the R archive, a minimum set of rules need nonetheless to be followed.

Although R can perform most of the analyses available in generalist software such as Stata, SPSS, or SAS, it has a broader potential since it can also be used for mapping, data mining or machine learning. Being a language also means that there are often several ways to carry out analyses in R, each one with its advantages and inconvenient. Users can also easily produce publication quality output from R thanks to its integration with the Markdown LaTeX document presentation system, and R graphs can also be imported into MS Word or LibreOffice documents.

By contrast with other statistical software, the R interface is rather minimal and consist merely of a terminal. In line with programming languages such a Python or C, R users tend to access it via an interface, or Integrated Development Environment (IDE). This guide uses the R Studio development environment, one of the most common IDE for R. The data used in this guide is the British Social Attitudes Survey, 2017, Environment and Politics: Open Access Teaching Dataset, which can be downloaded from the UK Data Service website without registration. The website also has instructions on how to acquire and download large-scale survey datasets. Links and further information about the other training resources available online are provided at the end of this document.

Although R has advantages over other statistical analysis software, it also has a few downsides, both of which are summarised below. Users should be reminded that as open-source software, R and its packages are developed by volunteers, which makes it a very flexible and dynamic project, but at the same time reliant on developers' free time and goodwill.

Table 1.1: Advantages and inconvenients of R

Pros	Cons
R is free and allows users to perform almost any analysis they want.	The learning curve may be steep for users who do not have a prior background in statistics or programming.

Pros	Cons
R puts statistical analysis closer to the reach of individual citizens rather than specialists.	
Transparency of use and programming of the software and its routines, which improves the peer-reviewing and quality control of the software in many cases.	
Very flexible.	Problem solving (for both advanced users and beginners) may be time-consuming, depending on how common the problem encountered, and may lead to more time spent solving technical rather than substantive issues.
Availability of a wide range of advanced techniques not provided in other statistical software	Many people who design R packages are, or will become busy academics. Packages can stop being maintained without notice.
A very large user base provides abundant documentation, tutorials, and web pages.	

There are several (sometimes many) ways of achieving a particular result in R. This can be confusing for novice researchers, but at the same time will allow users to tightly adjust their programmes to their needs.

2 Using R: essential information

2.1 Download and installation

R can be downloaded for free from the CRAN website and run like any other Windows application. Versions for Mac and Linux are also available. After installation, the standard and rather minimalist R interface that appears when the programme is launched is shown below.

```
library(foreign)
```

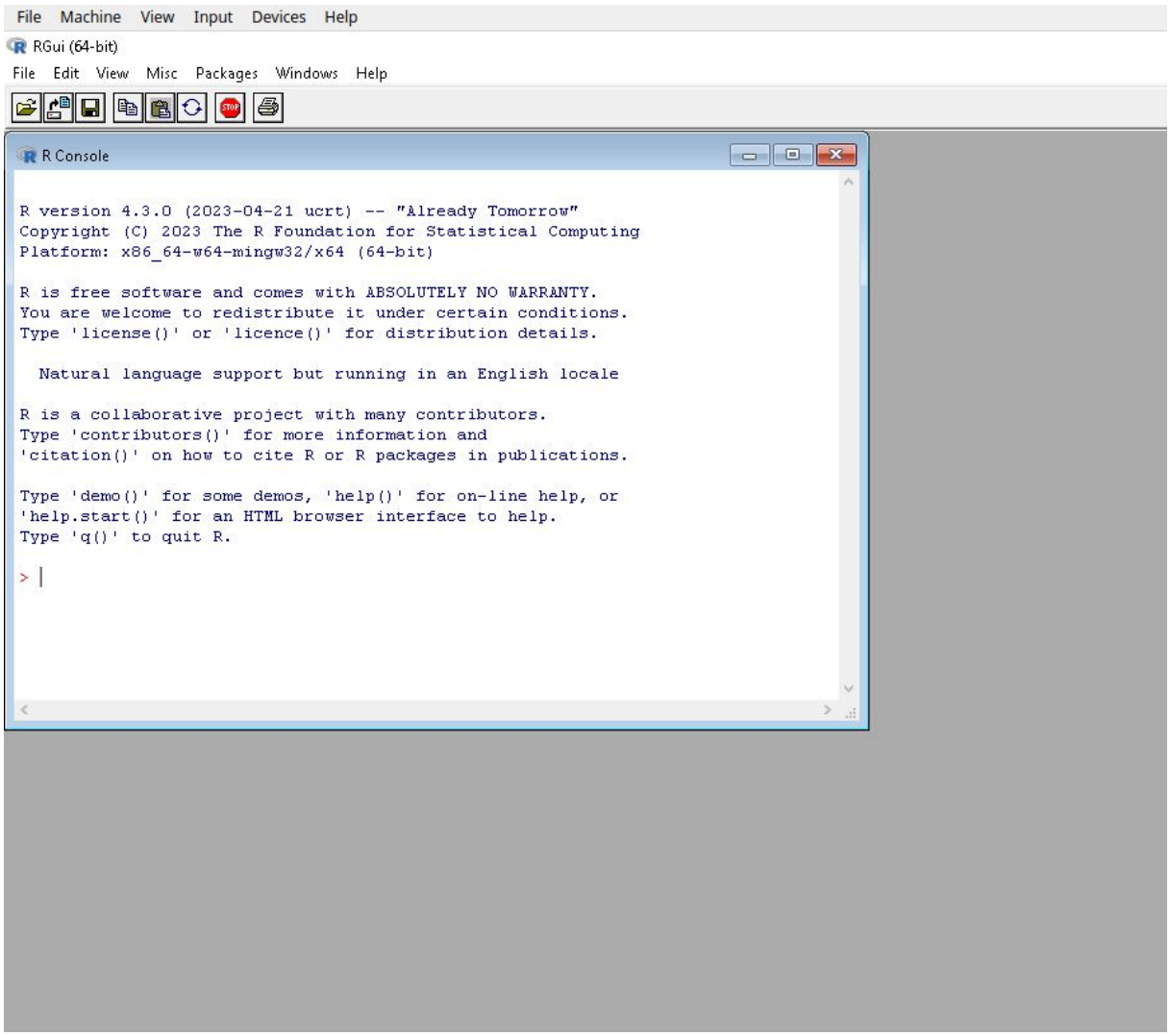


Figure 2.1: The standard R interface

This interface merely allows the user to type in commands one by one in the console, and to install packages via pull-down menus. However, this basic installation, although fully functional, is rather minimal, not very ergonomic or user friendly. As with other statistical software, the primary way of interacting with R for most is to write programs, even basic ones in a syntax file (also called script file) that is saved and run whenever needed, which is not directly feasible with the standard R GUI.

It is therefore highly recommended to use R via an Integrated Development Environment (ie a more sophisticated user interface) such as RStudio for beginners to intermediate users or the StatEt module for Eclipse for more advanced programmers. Both are free, available for Windows, MacOS and Linux and offer users a large number of additional functionalities, such as syntax highlighting, integration with Github. Given that it probably has the largest number of users RStudio will be used to demonstrate examplkes of R syntax in the remainder of this document. In order for this guide to remain as universal as possible, we will not rely on the advanced features of RStudio, instead using it merely as an interface to the R engine.

2.2 Installing and setting up RStudio

RStudio needs to be installed separately from R. The program can be downloaded from the RStudio website. The site will automatically generate a link to the version most compatible with the computer used to access it. Once downloaded double click on the file and follow the installation instructions.

By default, the R Studio interface consists of four main panels, respectively known as the script editor (top left panel), the console (bottom left panel), the Environment (top right panel) and the File/Directory/Help (bottom right panel).

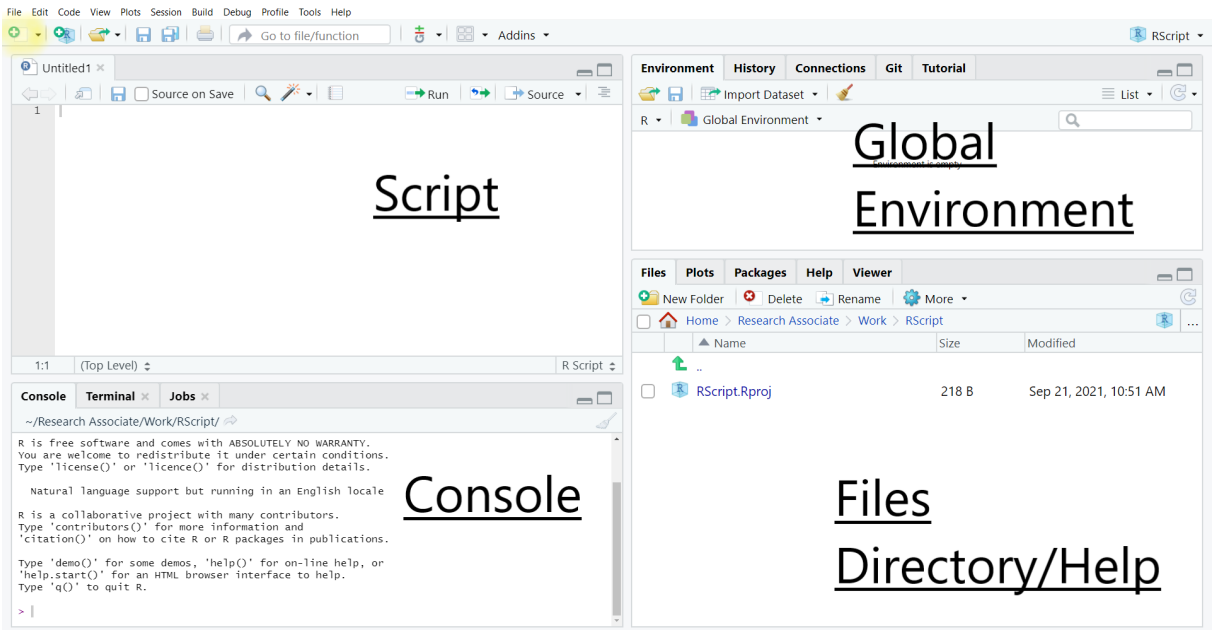


Figure 2.2: The R Studio default interface

As such a complex interface can be visually overwhelming for some users and is not required for the purpose of this guide, we will minimise the Global Environment and Files/Directory/Help

panels by clicking in the center of the window and dragging right to the edge of the screen. This way, only the script and console panels remain visible. The tiling of the panels can be customised in `Tools>Global Options>Pane Layout`. For instance, Script can be moved to the bottom of the window and Console to the top:

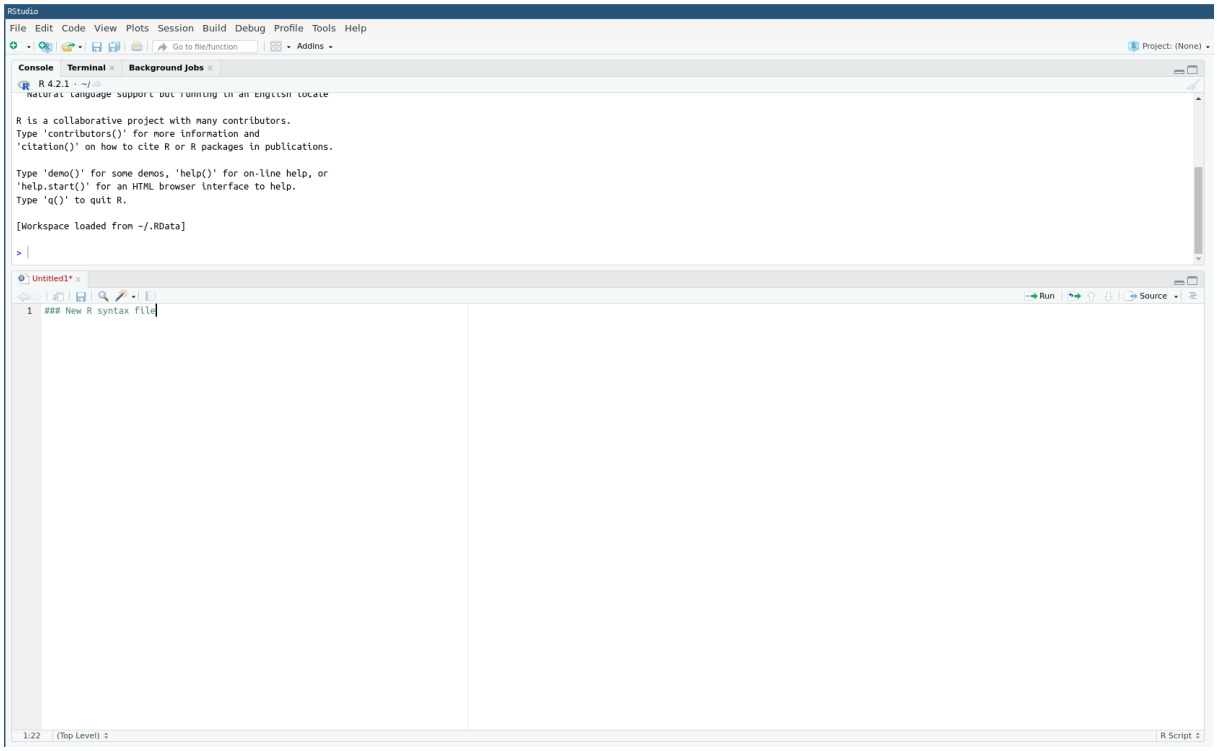


Figure 2.3: A customised R Studio interface

2.3 Interacting with R

As already mentioned, one can type R commands directly in the console of RStudio and/or by typing sequences of commands in a script file.

Most R commands adopt the following syntax:

```
> command(parameter1, parameter2, ...)
```

`All` R commands are followed by brackets, even if there are no parameters.

In the following example we are going to set up the default working directory, that is the default location for opening and storing files, by using the `getwd()` and `setwd()` commands. First, let us visualise the current default working directory.

```
getwd()
```

```
[1] "/home/piet/Dropbox/work/UKDS/RGuide/UKDS_RGuide"
```

Let us say we would like the code from this guide to be all in a folder called ‘R_UKDS’, to be located in ‘My Documents’. To tell R to use the folder ’R_UKDS, we can either create it from within Windows or ask R to do it for us. So type:

For Windows:

```
> setwd("C:/Documents and Settings/<INSERT YOUR USERNAME HERE>/My Documents/R_UKDS")
```

For Mac:

```
> setwd("/Users/<INSERT YOUR USERNAME HERE>/Documents/R_UKDS")
```

For Linux:

```
dir.create("~/Documents/R_UKDS")
setwd("~/Documents/R_UKDS")
```

Typing `getwd()` confirms that the change has been recorded.

```
getwd()
```

```
[1] "/home/piet/Dropbox/work/UKDS/RGuide/UKDS_RGuide"
```

Notes:

- Any character string that is neither a command or the name of an object (such as a variable name) needs to be put between inverted commas or quotation marks, otherwise it will be interpreted as the name of an object
- see the example below about loading user-created packages;
- Even when no parameters are specified for a command, brackets are compulsory as shown in the `getwd()` example above;
- R uses forward slashes rather than backslashes (unlike most other Windows applications) to separate directories. Using backlashes will return an error message;
- Although most R commands accept a large number of options to be specified, in many cases default values have been ‘factory set’ so that only the essential parameters need specifying.

The output of most R commands can be either directly displayed on the screen (as in the above example) or stored in objects that can be subsequently reused in further commands. This object-oriented feature separates R from traditional statistical software.

For instance, typing:

```
>   a<-getwd()
```

will store the output of `getwd()` (that is, the name of the current default directory) into an object called ‘a’. In order to view the content of a, one can just type its name:

```
>   a
```

Writing R scripts via R Studio {-} Most users will want to write their code in a script file, similar to the 'do' file in Stata or syntax file in SPSS. R script files end with the .R suffix. To open an existing R script in RStudio select `File>Open File` then the relevant script file. To create a new script select `File>New File>Open File` (shortcut: Control+Shift+N) this will open a new script window in which to type commands.

2.4 Installing and loading packages

Apart from a basic set of commands and functions, most of the tools offered by R are available in packages that are not provided during the main installation and need to be installed and downloaded separately from within R. For example, to install the 'foreign' package one need to type:

```
install.packages("foreign",repos = "https://cloud.r-project.org")
```

Installation only needs to be done once. If the address of the package repository is not specified via the `repos` option, a pull-down menu will appear, asking for one. Choosing `https://cloud.r-project.org` will automatically select the closest mirror site.

Originally, `Foreign` enabled users to import Stata (version 12 or older) or SPSS datasets. For Stata datasets saved under version 13 and above, the `haven` or `readstata13` package are required.

```
install.packages("haven",repos = "https://cloud.r-project.org")
```

To use a package already installed in the local R library, the `library()` command is needed:

```
library(foreign)
```

Simply typing:

```
> library()
```

Will list all libraries installed on the computer that can be loaded in memory. This can be a rather long list!

Besides a full archive of R packages, the CRAN website provides a series of manuals, including Writing R Extensions, which describes how users can write their own packages and submit them to CRAN.

Once a package is installed, it will be permanently stored in the local R library on the computer, unless deleted it with the `remove.packages()` command (not advised as this can break dependencies between packages!).

```
> remove.packages("name of the package")
```

Packages required for an analysis have to be loaded every time a new R session is started (But not every time a syntax file is run!).

2.5 Getting help

Within R, the most straightforward way to request help with a command consists of a question mark followed by the command name, without a space in between. The standard help system in R (unless using RStudio or Eclipse) relies on the default web browser installed on your computer (ie Chrome, Firefox or Edge in most cases) to display pages. Typing:

```
> ?getwd
```

Is equivalent of:

```
help('getwd')
```

and will open the help page for the `getwd()` command in the default web browser. If you are using RStudio or Eclipse, the help will most likely open in a new tab within the program.

This will work for any command directly available in the `Base` package that is loaded at startup or in other packages loaded via the `library()` command. Otherwise, R will return an error message.

Typing two question marks followed by a keyword will search all of R for the available documentation for that keyword:

```
??foreign
```

An index of all commands and functions in the foreign package can be obtained by typing:

```
help(package='foreign')
```

Note: this command only works because the 'foreign' package was previously loaded in memory with the `library()` command. More information about where to find help when using R is provided at the end of this document.

2.6 Objects

R is an object oriented language, which means that almost any information it uses is stored as 'objects' (i.e. containers) that can be manipulated independently. During an R session, multiple objects are available simultaneously (for instance datasets, but also summary tables or new variables produced from it). Typing:

```
> ls()
```

will list all the objects that are currently in memory.

Objects belong to `classes` or types which have distinct `properties`. There are many classes of objects in R. By comparison, Stata has only macros, variables and scalars that are directly available to most users. Common object classes include factors (these are equivalent to categorical variables), vectors (numerical variables – whether continuous or ordinal), data frames (datasets), matrices, etc. Not all operations are possible with all objects in R. More advanced users can also create their own object classes. Describing R objects and their properties is well beyond the purpose of this guide and users interested should consult the online documentation for further explanations.

To create or assign a value to an object, one uses the assignment operator (<-). For example:

```
> x <- 5
```

In this example we have assigned the value 5 to an object called x. If you type the letter x, the value '5' will be returned in your console. The object x will appear in the R environment after the ls() command.

```
x <- 5
x
```

```
[1] 5
```

```
ls()
```

```
[1] "has_annotations" "x"
```

Deleting objects

The rm() function can be used to remove objects from the environment (session). These objects can be variables, lists, datasets, etc. For instance, to remove the object 'x', or the fictitious dataset called 'mydata':

```
rm(x)
ls()
```

```
[1] "has_annotations"
```

```
> rm(mydata)
```

Data frames

Among the various classes of objects one may use in R, a few are essential to understand when analysing survey data. Their characteristics are briefly listed below;

Data frames are objects that come closest to datasets or excel sheets in traditional statistical software. They are objects that have indexed rows and columns, both of which may have names. Data frames columns can be seen as variables and lines or rows as observations. Each cell in the data frame can be uniquely identified by its position. Data frames are typically the object in which survey datasets are stored.

Let's assume that we have a small data frame called 'mydata'. Here are a few basic commands to examine it:

Determining the size of a data frame: the `dim()` command returns the number of rows and columns of a data frame

```
dim(mydata)
```

```
[1] 50  6
```

R tells us that our data is made of 50 rows and 6 columns, in other words of 50 observations and 6 variables. What if I want a quick overview of the dataset?

```
head(mydata)
```

```
    RSex skipmeal                   Married
1 Female       NA Married/living as married
2 Female        1        Separated/divorced
3 Female       NA Married/living as married
4   Male       NA             Never married
5   Male        1             Never married
6   Male       NA Married/living as married
         Poverty1                          HEdQual3
1        Was not Higher educ below degree/A level
2        Was not Higher educ below degree/A level
3        Was not             O level or equiv/CSE
4        Was not Higher educ below degree/A level
5        Was not                 No qualification
6 Was in poverty                             <NA>
  NatFrEst
1        5
2       30
3       50
4       50
5       50
6       10
```

The `head()` command displays the first six lines of the dataset. Depending on the number of variables the output of `head()` can become quickly overwhelming, as the size of the lines on most screens is limited!

Obtaining the names of variables (or columns) in the dataset: This can be done using either `ls()` which we already have used, or the `names()` commands. `ls()` returns the variables names, sorted alphabetically, whereas `names()` returns them in their actual order in the data frame.

```
ls(mydata)
```

```
[1] "HEdQual3" "Married"  "NatFrEst" "Poverty1"
[5] "RSex"     "skipmeal"
```

```
names(mydata)
```

```
[1] "RSex"     "skipmeal" "Married"  "Poverty1"
[5] "HEdQual3" "NatFrEst"
```

We can see that in the data frame, the “RSex” column comes in fact before “Poverty1”.

Accessing variables:

Each column of a data frame, or variable, can be accessed by its name preceded by the $ sign:

```
mydata$NatFrEst
```

```
 [1]  5 30 50 50 50 10  1 NA  5 50 60 25 30  3 25 10 80
[18]  5 50 50 40 45  2 82 60 40 10 10 40 15 30 30  2 10
[35] 75 99 70 50 10 30  1 10 85 45 70 25 40 30 10 25
attr(,"value.labels")
named numeric(0)
```

Alternatively, columns/variables and rows can be identified numerically by their position in the data frame using square brackets:

```
dataframe[row number,column number]
```

Given that `RSex` is the first column of our dataset

```
mydata[,1]
```

```
 [1] Female Female Female Male   Male   Male   Male
 [8] Female Male   Male   Male   Female Female Male
[15] Female Female Female Female Female Female Male
[22] Female Female Male   Male   Female Female Female
[29] Female Male   Female Female Female Female Male
[36] Male   Female Female Male   Female Male   Male
[43] Female Male   Female Male   Female Female Female
[50] Male
Levels: Male Female
```

Returns the same output as previously. Not specifying a row or column name within the square brackets tells R to display them all.

```
mydata[6,]
```

```
  RSex skipmeal                   Married
6 Male       NA Married/living as married
        Poverty1 HEdQual3 NatFrEst
6 Was in poverty     <NA>       10
```

Returns the values of all the variables for the sixth row of the data frame. Specifying both a row and column number, will return a unique observation:

```
mydata[6,6]
```

```
[1] 10
```

which in this case is 10. Finally, more than one column or row can be displayed by concatenating their number using the `c()` function:

```
mydata[c(6,9),c(1,6)]
```

```
  RSex NatFrEst
6 Male       10
9 Male        5
```

The above command returns respectively the sixth and 9th observations for the sixth column. Please note that columns names can also be used instead of their number, provided that they are put between inverted commas:

```
mydata[c(6,9),c('RSex','NatFrEst')]
```

```
  RSex NatFrEst
6 Male       10
9 Male        5
```

Returns the same result as previously. Having a data frame to hand allows us to explore other types of objects commonly found in R. The type of a variable can be displayed by simply using the `class()` function.

Numeric

`Numeric` objects are simple numerical vectors (ie a single or a list of numbers). Here this is the case for `NatFrEst`, the estimated proportion of people making wrong benefits claims, according to respondents to the survey.

```
class(mydata$NatFrEst)
```

```
[1] "numeric"
```

Character

Character objects are alphanumeric vectors, that is variables which consist of text string(s).

```
class(mydata$Married)
```

```
[1] "character"
```

Factors

An important feature of R is that categorical variables whether ordinal or polynomial are stored in objects known as **factors**. The main difference between factors and traditional categorical variables in Stata or SPSS is that they do **not** consist of discrete numerical values with which value labels are associated. They should be thought of instead as a special type of character variable with a discrete set of values, which are known as `levels`. In our data, `Rsex` (Gender of the respondent) is such an object:

```
class(mydata$RSex)
```

```
[1] "factor"
```

Let's further examine this factor.

```
levels(mydata$RSex)
```

```
[1] "Male"   "Female"
```

returns the levels (ie the values) of `RSex`. Even if 'Male' is the first level of `Rsex`, and female the second one, these do not correspond to underlying numbers in the data. Please also note that it is possible to change the ordering of factor levels with the `factor()` function. It is always a good idea to check the ordering of factor levels in a newly created variable.

```
mydata$RSex.New<-factor(mydata$RSex,levels = levels(mydata$RSex)[c(2,1)])
levels(mydata$RSex.New)
```

```
[1] "Female" "Male"
```

The above code tells R to create a new factor variable -`RSex.New` - whose levels are the same as the initial `RSex`, but with 'Female' coming first, and "Male", second. The name of the new variable is arbitrary.

3 Opening datasets in R

3.1 Essential information

In principle, any dataset whether in CSV, SPSS, SAS, or Stata format can be opened by R. There are a number of issues to consider however:

- The `foreign` package has been traditionally used to import SPSS and Stata datasets into R:
- its `read.spss()` and `write.spss()` functions respectively open and write .sav files. Given that both were developed from older versions of SPSS, it is therefore advised to check that their outcome is as expected. In addition:
- `read.spss()` does not store the data in a R data frame by default and will require the option `to.data.frame=T` to be specified.
- `read.spss()` may sometimes struggle with some numeric format and wrongly attempt to convert them as factor, which will result in error messages. It is therefore advised to limit the maximum number of levels that will be considered when converting factors by using the option `max.value.labels=`
- `read.dta()` and `write.dta()` respectively open and write Stata files up to version 12. An option to watch for is `convert.factor=T/F` which either will import Stata categorical variable as their underlying numeric value or instead will convert them into factors, using value labels as levels, which may be an issue for categorical variables with a large number of levels. Users have to bear in mind that the labels will by default sorted alphabetically.
- The `readstata13` package opens Stata datasets from version 13 onwards with the `read.dta13()` function and offers a more comprehensive set of options. `convert.factor=T/F` has the same effect as in `read.dta()` from `foreign`.
- Data frames created with either `read.dta()` or `read.dta13()` have extra information stored as attributes, which maybe useful to retrieve. For instance:

```
> mydata<-read.dta("Some_Stata_dataset.dta")
> attributes(mydata)$var.labels ### Retrieves the original Stata variable labels
```

- Finally the `haven` package opens SPSS, Stata and SAS files with respectively `read_spss()`, `read_dta()` and `read_sas()`. By contrast with the other two packages, it relies on ad hoc data formats and data structures for converting labelled categorical variables and attempts to mimic Stata's value and variable labels. More information is available here.

In order not to overcomplicate their initial exploration of R we recommend new users to use `read.spss()` or `read.dta13()` when importing datasets from either SPSS or Stata, rather than the more elaborate functions available in `haven`.

3.2 The 2017 British Social Attitudes Survey

For the rest of this guide, we will use the `British Social Attitudes Survey, 2017, Environment and Politics: Open Access Teaching Dataset`, which can be downloaded from the UK Data Service website. We will use the SPSS version of the dataset, which will be assumed to be saved in a `UKDS` folder created inside `Documents` folder. `C:\\Users\\Your_User_Name_Here\\Documents` We will set this as default working directory. This way, we won't have to specify the full path of files that we will be opening or saving.

We can finally open the file:

```
bsa<-read.spss(paste0(datadir,"8849spss_V1/bsa2017_open_enviropol.sav"),
               to.data.frame = TRUE,
               use.value.labels=TRUE,
               max.value.labels = 9)

dim(bsa)
```

```
[1] 3988   25
```

3.3 Understanding the dataset

As previously, we can find the number of observations and variables in the dataset by typing the following:

```
dim(bsa)
```

```
[1] 3988   25
```

We can see that there are 3,988 observations and 25 variables in the BSA dataset.

Typing:

```
ls()
```

```
[1] "bsa"             "datadir"         "has_annotations"
```

will show us that the object 'bsa' has appeared, but what if we want to get the list of all variables in the dataset? We need to type:

```
ls(bsa)
```

```
 [1] "actchar"          "actpol"           "carallow"         "carenvdc"
 [5] "carnod2"          "carreduc"         "cartaxhi"         "CCBELIEV"
 [9] "ChildHh"          "eq_inc_quintiles" "govnosa2"         "HEdQual3"
[13] "leftrigh"         "libauth"          "Married"          "PartyId2"
[17] "plnenvt"          "plnuppri"         "Politics"         "RAgeCat"
[21] "RClassGp"         "Rsex"             "Sserial"          "Voted"
[25] "WtFactor"
```

If we want to get a better sense of the data, we use the `head()` function which will return the first six rows.

```
head(bsa)
```

```
  Sserial   Rsex RAgeCat                   Married ChildHh
1  290001   Male   35-44 Married/living as married     Yes
2  290002 Female     65+        Separated/divorced      No
3  290003 Female   45-54 Married/living as married     Yes
4  290004 Female   45-54 Married/living as married     Yes
5  290005   Male     65+             Never married      No
6  290006 Female   25-34             Never married     Yes
                          HEdQual3                          eq_inc_quintiles
1             O level or equiv/CSE 3rd quintile - More than £292, up to £404
2 Higher educ below degree/A level                                      <NA>
3                           Degree 3rd quintile - More than £292, up to £404
4                           Degree 3rd quintile - More than £292, up to £404
5 Higher educ below degree/A level 4th quintile - More than £404, up to £577
6 Higher educ below degree/A level              Lowest quintile - Up to £175
                                 RClassGp
1 Lower supervisory & technical occupations
2        Semi-routine & routine occupations
3          Managerial & professional occups
4          Managerial & professional occups
5          Managerial & professional occups
6          Managerial & professional occups
                                                                                    CCBELIEV
1               I believe that climate change is taking place but not as a result of human actions
2 I believe that climate change is taking place and is, at least partly, a result of human actions
3                                                                                       <NA>
4                                                                                       <NA>
5                                                                                       <NA>
6                                                                                       <NA>
  carallow carreduc carnod2 cartaxhi carenvdc plnenvt plnuppri
1     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
2     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
3     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
4     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
5     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
6     <NA>     <NA>    <NA>     <NA>     <NA>    <NA>     <NA>
           Politics Voted               actchar                actpol
1             some,   Yes                 never                 never
2 ... a great deal,   Yes                  <NA>                  <NA>
3             some,   Yes                  <NA>                  <NA>
4 ... a great deal,   Yes once in the past year once in the past year
5    not very much,   Yes                  <NA>                  <NA>
6    not very much,   Yes                  <NA>                  <NA>
                  govnosa2         PartyId2 leftrigh  libauth  WtFactor
1                    agree           Labour      1.0 4.500000 0.9380587
2                 disagree           Labour      1.8 4.333333 0.6844214
```

```
3                       <NA>           Labour       NA       NA 0.9060821
4                      agree           Labour      1.5 2.500000 1.3085513
5                       <NA> Liberal Democrat      2.0 3.166667 0.4392823
6 neither agree nor disagree             <NA>      3.0 3.000000 0.5695720
```

Single variables for example, `Rsex` (gender of respondents) may be also summarised with `head()`, which returns as previously the first six observations of the gender variable, whereas typing

```
head(bsa$Rsex)
```

```
[1] Male   Female Female Female Male   Female
Levels: Male Female
```

Simply typing the name of a variable as in:

```
bsa$Rsex
```

will list the first 1000 observations of the variable. Other commands provide more useful information, such as `summary()`.

```
summary(bsa$Rsex)
```

```
  Male Female
  1806   2182
```

Summary is a generic function that tailors the most appropriate output to an object class. As Rsex is a categorical variable. The output of summary() is identical to what we would have obtained with the default tabulation fuction `table()`:

```
table(bsa$Rsex)
```

```
  Male Female
  1806   2182
```

When encountering a continuous variable, `summary()` will compute basic descriptive statistics (mean, median, quartiles, maximum and minimum). For example, in the case of the libertarian-authoritarian scale `libauth`:

```
summary(bsa$libauth)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.167   3.000   3.500   3.511   4.000   5.000     775
```

3.4 Identifying and selecting variables

As we have already seen, variables are objects. R automatically stores variables using the appropriate object class. Categorical variables are 'Factors' with 'Levels' as categories within these, while continuous variables are 'Numeric' types of objects. The `class()` displays the type of an object:.

```
class(bsa$Rsex)
```

```
[1] "factor"
```

The levels() function returns the categories of the variable.

```
levels(bsa$Rsex)
```

```
[1] "Male"   "Female"
```

4 Essentials of Data Manipulation

In this section we will cover how to recode variables and deal with missing data.

4.1 Creating and transforming numerical variables

Let's say we would like to transform our numerical political orientation variable: `leftrigh` into a logarithmic scale. We can use the `log()` function which is directly available in R Base package and simply returns the natural logarithm (base-e). We will use the assignment operator (<-) to create a new variable called 'leftright_log' from the original variable

```
bsa$lnleftrigh_log <- log(bsa$leftrigh)
```

Note that if we had not specified `bsa$` the command would have created a transformed variable outside of the BSA data frame. We can now check the results with `summary()`

```
summary(cbind(bsa$lnleftrigh,bsa$leftrigh))
```

```
       V1               V2
 Min.   :0.0000   Min.   :1.00
 1st Qu.:0.6931   1st Qu.:2.00
 Median :0.8755   Median :2.40
 Mean   :0.8707   Mean   :2.52
 3rd Qu.:1.0986   3rd Qu.:3.00
 Max.   :1.6094   Max.   :5.00
 NA's   :782      NA's   :782
```

Please note that it is not possible to pass several variables names directly to `summary()`. We need to group them first into a temporary object using `cbind()`. In the output `V1` refers to the first variable, `lnleftrigh`.

What if we - hypothetically - wanted to do the same with level of agreement to the statement "People like me don`t have any say about what the government does", a categorical variable, originally ranging from 1 (Agree strongly) to 5 (Disagree strongly')? If we try to repeat what we did before:

```
bsa$lngovnosa2 <- log(bsa$govnosa2)
```

```
Error in Math.factor(bsa$govnosa2): 'log' not meaningful for factors
```

… we get an error, due to the fact that `govnosa2` was imported as a factor, and that `log()` can only be applied to numeric objects.

```
class(bsa$govnosa2)       #check to see if it is numeric
```

```
[1] "factor"
```

```
bsa$govnosa2.n <- as.numeric(bsa$govnosa2)#convert to numeric
class(bsa$govnosa2.n)       #check again
```

```
[1] "numeric"
```

```
bsa$lngovnosa2 <- log(bsa$govnosa2.n) #create new log of the leftright      variable
summary(cbind(bsa$lngovnosa2,bsa$govnosa2.n))
```

```
       V1               V2
 Min.   :0.0000   Min.   :1.000
 1st Qu.:0.6931   1st Qu.:2.000
 Median :1.0986   Median :3.000
 Mean   :0.8910   Mean   :2.705
 3rd Qu.:1.3863   3rd Qu.:4.000
 Max.   :1.6094   Max.   :5.000
 NA's   :2447     NA's   :2447
```

We can also create a completely new variable in the dataset. For instance, the following will create `test` with a constant value of 1.

```
bsa$test <- 1
```

4.2 Categorical variables

We saw in Section 3 that categorical variables are objects called **factors** in R, with a fixed set of possible numerical or alphanumerical values (levels) which can be accessed with the `levels()` function.

```
levels(bsa$Married)
```

```
[1] "Married/living as married" "Separated/divorced"
[3] "Widowed"                   "Never married"
```

The number in the output does not refer to underlying numerical values to which labels are added as with other statistical packages, but instead to the position of a given level in the list returned by `level()`.

4.3 Recoding variables

The following commands will create a new variable called `Married2` where respondents are categorised into two new categories: 'Not partnered' and 'Partnered'. The "separated/divorced" and "Never married" categories of the "Married" variable are recoded as "Not partnered". It is always advised to create new variables when recoding old ones so the original data is not tampered with.

```
bsa$Married2 <- ifelse(bsa$Married=="Married/living as married","Partnered",bsa$Married)
bsa$Married2 <- ifelse(bsa$Married=="Widowed" |
                       bsa$Married=="Never married" |
                      bsa$Married=="Separated/divorced","Not partnered",bsa$Married2)

table(bsa$Married2)
```

```
Not partnered     Partnered
         1778          2209
```

The second and fourth categories have been renamed to 'Not partnered'. Now we have two levels * Partnered *Not partnered

`ifelse()` is a convenient tool to use when it is required to work with Base R or when the variables have a limited number of categories. More complex cases may require a more advanced function. The `dplyr` library provides a comprehensive set of data manipulation tools.

```
library(dplyr)
bsa<-bsa%>%
     mutate(Married3=case_when(
               Married == "Married/living as married" ~ "Partnered",
               Married == "Separated/divorced" | Married == "Widowed" ~ "Not Partnered",
               Married == "Never married" ~ "Not Partnered"
)
)
bsa$Married3<-as.factor(bsa$Married3)
summary(bsa$Married3)
```

```
Not Partnered     Partnered          NA's
         1778          2209             1
```

We just created the `Married3` variable, which is identical to `Married2` above, but using the more powerful syntax made available by `dplyr`. Let's decompose it:

- `dplyr` use the `pipe` symbol ie `%>%` which enables to sequentially combine functions. We will come back to this later in this guide.
- `mutate()` is the generic variable creation/alteration command, and can handle complex combinations of conditions as well as multiple simultaneous variable creation operations.

- `case_when()` is the function that allows recoding numerical, character, or factor variables. On the left hand side of the tilde `~` are the condition or the values that need to be matched in the original variable , and on the right hand side, the attributed ie recoded values in the new variable. Note that in this case, the recoded variable is by default a character object and needs to be converted into a factor for easier manipulation.

Extra tips:

- As with any data manipulation exercise, caution is required, and it is recommended to create new variables with the recoded value rather than alter an original variable when handling missing values.
- The standard value attribution command in R is `<-`. However, `=` will also work in many cases.
- Unless explicitly specified (in our case, by adding the bsa$ prefix to variable name), the objects created are not included in the data frame from which they were computed.

4.4 Missing Values

Explicit missing values in R (ie values that R itself considers as missing) are represented as `NA` for factors and numerical variables. For character variables, missing values are simply empty strings, ie `""`. R has a series of functions specifically designed to handle NAs.

R has fewer safety nets than other packages for handling missing values. Most function won't issue warn users about whether or how many how many observations with missing values have been dropped. On the other hand, some commands will return error messages and won't run when missing values are present. This is the case of `mean()` for example.

4.4.1 Inspecting missing data

The logical function `is.na()` assesses each observation in variables and identifies whether cases are valid or missing. The result will appear as a boolean TRUE/FALSE vector for each observation. `is.na()` can be combined with other functions:

- With `table()` in order to get the frequencies of missing values of a specific variable.
- With sum() in order to count the number of missing observations of variables or whole datasets.

```
table(is.na(bsa$leftrigh))#of missing values in the leftright variable
```

```
FALSE  TRUE
 3206   782
```

```
sum(is.na(bsa)) # of missing values in the whole dataset
```

```
[1] 41486
```

```
mean(is.na(bsa$leftrigh)) #returns the proportion of NAs...
```

```
[1] 0.1960883
```

```
mean(is.na(bsa)) # returns the proportion in the dataset
```

```
[1] 0.3355712
```

4.4.2 Recoding missing values as NA (continuous variables)

It may sometimes be useful to recode implicit missing values (ie considered by the data producer as missing, but not by R) of either numeric objects or factors into `<NA>`, in order to simplify case selection when conducting analyses. This can either be done with Base R code or the more advanced data manipulation functions from the dplyr package that we explored earlier.

Let's assume for a moment that we would like to get rid of respondents aged under 25 for our analysis. A safe way to proceed is by creating a new variable.

```
bsa$RAgeCat2 <- bsa$RAgeCat #duplicate variable
table(bsa$RAgeCat2)
```

```
18-24 25-34 35-44 45-54 55-59 60-64   65+
  223   591   650   729   320   333  1138
```

```
bsa$RAgeCat2[bsa$RAgeCat2=="18-24" ] <- NA #convert responses "18-24"  to NA
table(bsa$RAgeCat2)
```

```
18-24 25-34 35-44 45-54 55-59 60-64   65+
    0   591   650   729   320   333  1138
```

```
table(is.na(bsa$RAgeCat2))
```

```
FALSE  TRUE
 3761   227
```

Why is the number of missing values 227 and not 223 as the original number of respondents aged 18-24? Because there were already 3 missing values for the RAgeCat variable.

We can also notice that although there are now no observation left in the 18-24 category, it is still displayed by `table()`. This is because levels are attributes of factors and are not deleted with observations. We can remove unused levels permanently with `droplevels()`

```
bsa$RAgeCat2<-droplevels(bsa$RAgeCat2)
table(bsa$RAgeCat2)
```

```
25-34 35-44 45-54 55-59 60-64   65+
  591   650   729   320   333  1138
```

4.4.3 Working with missing values

Explicit missing values (coded as NA) can be taken care of by R's own missing values functions. For instance using the `na.rm=T` or `na.rm=TRUE` option will remove missing values from an analysis (typing `?na.rm` will provide more information). Below is a summary of how NAs are dealt with by common R commands:

Table 4.1: Treatment of missing values by R commands

Command	Default action	Parameter
mean(), sd(),median()	Includes NA (may return an error)	na.rm=T
cor(),cov()	Includes NA (may return an error)	*use="complete.obs"*
table()	Excludes NA	*useNA = "always"* to display NAs
xtabs()	Excludes NA	*addNA = T* to display NAs
lm(),glm()	Excludes NA	*na.action=NULL*

4.5 Subsetting datasets

When analysis survey data. it is often necessary to limit the scope of computation to specific groups or subset of the data we may be interested in. There are many ways of subsetting datasets in R. We will review the most common here.

Using Base R

Most subsetting commands involve some form of conditions whereby the characteristics of a subsample of interest are specified. Suppose we would like to examine the interest for politics among people aged 18-24.

We can either create an adhoc data frame:

```
table(bsa$Politics)
```

```
... a great deal,      quite a lot,             some,    not very much,
              739               982              1179               708
 or, none at all?
              379
```

```
bsa.young<-bsa[bsa$RAgeCat=="18-24",]
table(bsa.young$Politics)
```

```
... a great deal,      quite a lot,             some,    not very much,
               29                41                72                56
 or, none at all?
               25
```

```
bsa.young<-bsa[bsa$RAgeCat=="18-24",]
```

Or we can directly limit the extent of the analysis on the go:

```
  table(bsa$Politics[bsa$RAgeCat=="18-24"])
```

```
... a great deal,      quite a lot,             some,    not very much,
               29                41                72                56
 or, none at all?
               25
```

In the first example it was necessary to include a comma after specg the condition. This is meant to indicate that we want to retain all variables ie columns in the dataset. The comma is not necessary in the second example as we are already working with a single variable.

Using dplyr*

Now suppose we want to further restrict the analysis to people identifying as Males. We could use the same Base R syntax as above, but with more than one condition coding tends to become cumbersome. We could instead use the more convenient syntax from the `dplyr` package. Either:

```
bsa.young.males<-bsa%>%filter(RAgeCat=="18-24" & Rsex=="Male")
table(bsa.young.males$Politics)
```

```
... a great deal,      quite a lot,             some,    not very much,
               15                22                30                18
 or, none at all?
                9
```

Or. as before embed it as a condition within `table()` :

```
  table(bsa%>%filter(RAgeCat=="18-24" & Rsex=="Male")%>%select(Politics) )
```

```
Politics
... a great deal,      quite a lot,             some,    not very much,
               15                22                30                18
 or, none at all?
                9
```

We are now equipped with the necessary information to move to the next stage and carry out basic analysis using R.

5 Descriptive statistics

```
library(dplyr)
library(Hmisc)
library(ggplot2)
library(haven)
library(foreign)
datadir<-"/home/piet/Dropbox/work/UKDS/rguide/data/"
bsa<-read.spss(paste0(datadir,"8849spss_V1/bsa2017_open_enviropol.sav"),
               to.data.frame = TRUE,
               use.value.labels=TRUE,
               max.value.labels = 9)
```

5.1 Continuous variables

Producing descriptive statistics in R is relatively straightforward, as key functions are included by default in the Base package. We have already seen above that the `summary()` command provides essential information about a variable. For instance,

```
summary(bsa$leftrigh)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00    2.00    2.40    2.52    3.00    5.00     782
```

provides information about the mean, median and quartiles of the political scale of respondents.

The `describe()` command from the `Hmisc` package provides a more detailed set of summary statistics.

```
library(Hmisc)
describe(bsa$leftrigh)
```

```
bsa$leftrigh
       n  missing distinct     Info     Mean      Gmd      .05      .10
    3206      782       30    0.993     2.52   0.8831      1.2      1.4
     .25      .50      .75      .90      .95
     2.0      2.4      3.0      3.6      4.0

lowest : 1    1.2  1.4  1.5  1.6 , highest: 4.4  4.6  4.75 4.8  5
```

`describe()` also provides the number of observations (including missing and unique observations), deciles as well as the five largest and smallest values.

Commands producing single statistics are also available:

```
mean(bsa$leftrigh, na.rm = T)
```

```
[1] 2.519911
```

```
sd(bsa$leftrigh, na.rm = T)
```

```
[1] 0.7852958
```

```
median(bsa$leftrigh, na.rm = T)
```

```
[1] 2.4
```

```
max(bsa$leftrigh, na.rm = T)
```

```
[1] 5
```

```
min(bsa$leftrigh, na.rm = T)
```

```
[1] 1
```

We could combine the output from the above commands into a single line using the `c()` function:

```
c(
  mean(bsa$leftrigh, na.rm = T),
  sd(bsa$leftrigh, na.rm = T),
  median(bsa$leftrigh, na.rm = T),
  max(bsa$leftrigh, na.rm = T),
  min(bsa$leftrigh, na.rm = T)
)
```

```
[1] 2.5199106 0.7852958 2.4000000 5.0000000 1.0000000
```

As we saw previously, the `na.rm = T` option prevents missing values from being taken into account (in which case the output would have been NA, as this is the default behaviour of these functions). Using these individual commands may come in handy, for instance when further processing of the result is needed:

```
m <- mean(bsa$leftrigh, na.rm= T)
```

Let's round the results to two decimal places:

```
rm <- round(m,2)
```

We can see the final results by typing:

```
rm
```

```
[1] 2.52
```

Note:

```
round(mean(bsa$leftrigh,na.rm=T),2)
```

```
[1] 2.52
```

would produce the same results using just one line of code .

5.2 Bivariate association between continuous variables

R provides a wide range of bivariate statistics under its base packages. The cor() and cov() functions provide basic measures of association between two variables. For instance, in order to measure the correlation between the leftright scale and the libertarian-authoritarian scale: The later variable is a numeric variable that details how far someone sits on the libartrian – authoritarian scale from 1 to 5

```
cor(bsa$leftrigh, bsa$libauth, use='complete.obs')
```

```
[1] 0.009625928
```

A correlation of 0.009 indicates an positive but very small relationship. It can be translated to mean 'an increase in authoritarianism is associated with a marginal increase in rightwing views.

Note: When using the cor() and cov() functions missing values are dealt with the 'use=' "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs" options. See '?cor' for additional information.

5.3 Categorical Variables

As with continuous variables, R offers several tools that can be used to describe the distribution of categorical variables. One- and two-way contingency tables are the most commonly used.

5.3.1 One way frequency tables

There are several R commands that we can use to create frequency tables. The most common ones `table()`,`xtabs()` or `ftable()` which return the frequencies of observations within each level of a factor. For example, in order to obtain the political affiliation of BSA respondents in 2017:

```
table(bsa$PartyId2)
```

```
    Conservative           Labour Liberal Democrat      Other party
            1263             1479              241              193
            None      Green Party
             515               79
```

As with any other R functions, the outcome of `table()` can be stored as an object for further processing:

```
a<-table(bsa$PartyId2)
```

`table()` does not compute proportions or percentages. Proportions are obtained using the `prop.table()` function which in turn does not produce percentages. It is also a good idea to round the results for greater readability. Either:

```
round(100*prop.table(a),1)
```

```
    Conservative           Labour Liberal Democrat      Other party
            33.5             39.2              6.4              5.1
            None      Green Party
            13.7              2.1
```

… or:

```
round(100*
        prop.table(
          table(bsa$PartyId2)
        ),
      1)
```

```
    Conservative           Labour Liberal Democrat      Other party
            33.5             39.2              6.4              5.1
            None      Green Party
            13.7              2.1
```

5.3.2 Two way or more contingency table

The simplest way to produce a two-way contingency table is to pass another variable to `table()`:

```
table(bsa$PartyId2, bsa$Rsex)
```

```
                  Male Female
  Conservative      627    636
  Labour            644    835
  Liberal Democrat  124    117
  Other party        97     96
  None              199    316
  Green Party        31     48
```

However, when dealing with more than one variable it is recommended to use xtabs() instead as it has a number of desirable functions directly available as option. The syntax is slightly different as it relies on a formula ie a R object consisting of elements separated by a tilde '~'. The variables to be tabulated are specified on the right hand side of the formula.

```
xtabs(~PartyId2 +Rsex,
      data = bsa)
```

```
                 Rsex
PartyId2          Male Female
  Conservative      627    636
  Labour            644    835
  Liberal Democrat  124    117
  Other party        97     96
  None              199    316
  Green Party        31     48
```

The `data=` parameter does not have to be explicitly specified as simply using ´`bsa`' will work. Other useful options are:

- subset=, which allows direct specification of a subpopulation from which to derive the table;
- drop.unused.levels=T to remove empty levels (categories with zero observations) from being displayed;
- weights~ variables on the right hand side of the formula will be treated as weights, a useful feature for survey analysis.

As previously `prop.table()` is necessary in order to obtain proportions:

```
b<-xtabs(~PartyId2 +Rsex,bsa)
round(100*prop.table(b),1) ### Cell percentages
```

```
                 Rsex
PartyId2          Male Female
  Conservative    16.6   16.9
  Labour          17.1   22.1
  Liberal Democrat  3.3    3.1
  Other party       2.6    2.5
  None              5.3    8.4
  Green Party       0.8    1.3
```

The largest group in the sample (22.1%) is made of labour-voting females, the smallest of green-voting males.

```
round(100*prop.table(b,1),1) ### Option 1 for row percentages
```

```
                Rsex
PartyId2         Male Female
  Conservative   49.6   50.4
  Labour         43.5   56.5
  Liberal Democrat 51.5   48.5
  Other party    50.3   49.7
  None           38.6   61.4
  Green Party    39.2   60.8
```

Conservative voters are more or less evenly split between men and women, whereas Labour and Green voters are more likely to be women.

```
round(100*prop.table(b,2),1) ### Option 2 for column percentages
```

```
                Rsex
PartyId2         Male Female
  Conservative   36.4   31.1
  Labour         37.4   40.8
  Liberal Democrat  7.2    5.7
  Other party    5.6    4.7
  None           11.6   15.4
  Green Party    1.8    2.3
```

Similar proportions of men voted Conservative and Labour (36-37%), whereas women were clearly more likely to vote Labour.

For some reason, there is not a straightforward way to obtain percentages in three-way contingency tables with either `xtabs()` or `table()`. This is where `ftable()` comes handy.

```
round(100*prop.table(
  ftable(RAgeCat~PartyId2 +Rsex,data=bsa)
  ,1),1) ### Option 2 for column percentages
```

```
                      RAgeCat 18-24 25-34 35-44 45-54 55-59 60-64  65+
PartyId2         Rsex
Conservative     Male             3.0   7.8  13.9  15.0   8.6   9.3 42.4
                 Female           2.7   7.1   8.8  18.6   8.8   8.7 45.4
Labour           Male             7.6  16.1  14.3  21.3   7.5   9.3 23.9
                 Female           7.9  20.2  19.2  18.8   7.1   7.1 19.7
Liberal Democrat Male             0.8  13.7  19.4  15.3   9.7   8.9 32.3
                 Female           4.3   9.4  26.5   6.0   6.0   9.4 38.5
Other party      Male             3.1  14.4  11.3  17.5  11.3  16.5 25.8
                 Female           5.2  14.6  15.6  16.7  11.5   8.3 28.1
None             Male             7.5  22.1  20.6  17.1  11.1   6.0 15.6
```

```
                Female           8.5  22.5  20.6  21.8   7.3   6.3 13.0
Green Party     Male             6.5  32.3  16.1  29.0   6.5   3.2  6.5
                Female           6.2  18.8  25.0  20.8   6.2  10.4 12.5
```

The tables gives the relative age breakdown for each gender/political affiliation combination (ie row percentages).

5.4 Grouped summary statistics for continuous variables

A common requirement in survey analysis consist in being able to compare descriptive statistics across subgroups of the data. There are different ways to do this in R. We demonstrate below the most straightforward one, which is obtained by using some of the functions available in the `dplyr` package.

```
bsa%>%group_by(PartyId2)%>%summarise(
mdscore=median(libauth,na.rm=T),
sdscore=sd(libauth,na.rm=T))
```

```
# A tibble: 7 x 3
  PartyId2         mdscore sdscore
  <fct>              <dbl>   <dbl>
1 Conservative        3.67   0.587
2 Labour              3.33   0.774
3 Liberal Democrat    3.17   0.726
4 Other party         3.67   0.739
5 None                3.67   0.584
6 Green Party         2.83   0.872
7 <NA>                3.67   0.564
```

The above command produces a table of summary values (median and standard deviations) of the Liberal vs authoritarian scale. We can see from the first one that Green party voters are the most liberal, followed by Labour, whereas non voters and Conservatives are the most authoritarian. Liberal Democrats are the most cohesive group (ie with the smallest standard deviation).

```
bsa%>%group_by(Rsex,PartyId2)%>%summarise(
mnscore=sd(libauth,na.rm=T) ,mdscore=median(libauth,na.rm=T))
```

```
# A tibble: 14 x 4
# Groups:   Rsex [2]
   Rsex   PartyId2         mnscore mdscore
   <fct>  <fct>               <dbl>   <dbl>
 1 Male   Conservative        0.607    3.67
 2 Male   Labour              0.765    3.33
 3 Male   Liberal Democrat    0.766    3.17
 4 Male   Other party         0.703    3.83
 5 Male   None                0.616    3.67
```

```
 6 Male   Green Party        1.04     2.67
 7 Male   <NA>               0.603    3.67
 8 Female Conservative       0.565    3.67
 9 Female Labour             0.781    3.33
10 Female Liberal Democrat   0.688    3.17
11 Female Other party        0.773    3.67
12 Female None               0.565    3.67
13 Female Green Party        0.744    3
14 Female <NA>               0.540    3.67
```

When further broken down by gender, we can see that overall the same trends remain valid, with some nuances: male Green supporters are markedly more liberal than their female counterpart, the opposite being true among Conservative supporters.

Instead of tables of summary statistics, we may want to have summary statistics computed as variables that will be added to the current dataset for each corresponding gender/political affiliation group. This is straightforward to do with dplyr, we just need to use the `mutate()` command.

```
bsa<-
bsa%>%group_by(Rsex,PartyId2)%>%mutate(
msscore=sd(libauth,na.rm=T) ,mdscore=median(libauth,na.rm=T))
```

However, we also need to add the newly created variables into the existing bsa dataset, which the first line of the syntax above does. We can check that the variables have been created and that the correct values have been allocated to each sex/affiliation category.

```
names(bsa)
```

```
 [1] "Sserial"          "Rsex"             "RAgeCat"          "Married"
 [5] "ChildHh"          "HEdQual3"         "eq_inc_quintiles" "RClassGp"
 [9] "CCBELIEV"         "carallow"         "carreduc"         "carnod2"
[13] "cartaxhi"         "carenvdc"         "plnenvt"          "plnuppri"
[17] "Politics"         "Voted"            "actchar"          "actpol"
[21] "govnosa2"         "PartyId2"         "leftrigh"         "libauth"
[25] "WtFactor"         "msscore"          "mdscore"
```

```
bsa[4:8,c("Rsex","PartyId2","mdscore")]
```

```
# A tibble: 5 x 3
# Groups:   Rsex, PartyId2 [4]
  Rsex   PartyId2         mdscore
  <fct>  <fct>              <dbl>
1 Female Labour              3.33
2 Male   Liberal Democrat    3.17
3 Female <NA>                3.67
4 Male   Liberal Democrat    3.17
5 Female Green Party         3
```

6 Producing weighted estimates

Most users of social surveys are interested at some point in inferring nationally representative estimates and/or compensate for bias involved in the sampling process when conducting analyses: sampling and non-response bias. These are often tackled with sampling weights, which are meant to correct estimates for the under/over representation of certain groups in the sample and adjusts standard errors accordingly.

However, robust inference usually relies not just on weighting estimates but also on factoring in the survey design when conducting analyses – which can be done with the `survey` package in R, but is a topic that goes beyond the present guide. At the same time for users who are concerned with reducing bias rather than producing publication-quality estimates, it may be useful to be aware how common R commands and operations can be used with weights.

Some of the most common ones are mentioned below:

Central tendency and dispersion (continuous variables)

The `stats` packages which comes with the installation of Base R includes `weighted.mean()` which, as indicated by its name, computes weighted estimates of the mean of a variable when weights are provided. However the Hmisc package includes a more comprehensive set of functions that can be used when weighting estimates. The code below provides an illustration of weighted means, variance and median of the left-right score used before, each time comparing it with the unweighted estimate:

```
### Mean
c(mean(bsa$leftrigh,na.rm=T),wtd.mean(bsa$leftrigh,bsa$WtFactor))
```

```
[1] 2.519911 2.521589
```

```
### Variance
c(var(bsa$leftrigh,na.rm=T),wtd.var(bsa$leftrigh,bsa$WtFactor))
```

```
[1] 0.6166894 0.6195378
```

```
### Median and quartiles
c(quantile(bsa$leftrigh,na.rm=T,probs=c(.25,.5,.75)),
  wtd.quantile(bsa$leftrigh,bsa$WtFactor,probs=c(.25,.5,.75)))
```

```
25% 50% 75% 25% 50% 75%
2.0 2.4 3.0 2.0 2.4 3.0
```

The above functions can be used in conjunction with `group_by()` and `summarise()` in order to compute weighted estimates of continuous variables by groups of categorical variables:

```
bsa%>%
  filter(!is.na(RAgeCat))%>%group_by(RAgeCat)%>%
  summarise(Mean=wtd.mean(leftrigh,WtFactor),
            Var=wtd.var(leftrigh,WtFactor),
            Median=wtd.quantile(leftrigh,WtFactor,probs=c(.5)))
```

```
# A tibble: 7 x 4
  RAgeCat  Mean   Var Median
  <fct>   <dbl> <dbl>  <dbl>
1 18-24    2.49 0.556    2.4
2 25-34    2.56 0.577    2.6
3 35-44    2.52 0.615    2.4
4 45-54    2.53 0.671    2.6
5 55-59    2.54 0.653    2.4
6 60-64    2.46 0.685    2.4
7 65+      2.52 0.613    2.4
```

6.1 Frequencies and contingency tables

Neither `ftable()` or `table()` used above allow for using weights. And although the `Hmisc` packages includes the `wtd.table()` function for single frequency tables, we recommend using `xtabs()` as previously, as it it more versatile:

```
## Unweighted vs weighted frequency tables
cbind(Unweighted=round(100*prop.table(xtabs(~plnenvt,bsa)),1),
      Weighted=round(100*prop.table(xtabs(WtFactor~plnenvt,bsa)),1)
)
```

```
                          Unweighted Weighted
agree strongly                   4.6      4.8
agree                           16.0     15.9
neither agree nor disagree      33.2     33.2
disagree                        36.3     37.0
disagree strongly               10.0      9.0
```

Weights are passed to `xtabs()` by specifying their name on the left hand side of the equation (or the tilde `~`)

Obtaining weighted contingency tables follow the same logic:

```
## Unweighted vs weighted contingency tables
cbind(round(100*prop.table(xtabs(~plnenvt+Rsex,bsa),1),1),
      round(100*prop.table(xtabs(WtFactor~plnenvt+Rsex,bsa),1),1)
)
```

```
                           Male Female Male Female
agree strongly             50.0   50.0 46.1   53.9
agree                      47.2   52.8 53.5   46.5
neither agree nor disagree 40.8   59.2 43.6   56.4
disagree                   47.9   52.1 52.7   47.3
disagree strongly          42.3   57.7 41.5   58.5
```

6.2 Robust inference

The weighting procedures described above could be described as 'quick and dirty' in that they mostly compute point estimates – ie a single value – and do not provide a reliable idea of their precision. Computing the precision of survey data estimates – usually via their standard error – usually requires more than just adding weights to a command. Information about the survey design, its primary sampling units, strata and clusters is requires so that robust standard errors, statistical tests and/or confidence interval are computed.

The `Survey` package was designed in order to deal with this set of issues. It provides functions for integrating survey design into R as well as computing common estimates. We describe below the most important features. In order to use survey fonctions consist one first needs to create a svydesign object, in essence a version of the data that incorporates the sample design information available, then to compute the estimate using the svydesign object.

An common issue with survey datasets available in the UK is that sampling information is often only available in secure lab version of the data, restricting its access to authorised users. Although it is sometimes possible to use available variables to account for aspects of the sample design – region as a strata in the case of stratified samples – in most cases users are left with computing standard errors without sample design information, which amounts to assuming that the sample was drawn purely at random. Even if this is the case however, using the `survey` package is recommended, as it provides a coherent framework for computing survey parameters.

```
library(survey) ### Loading the package in memory
bsa.design<-svydesign(ids =~1,weights=~WtFactor,data=bsa)
```

The code above simply declares the survey design by creating the `bsa.design` object (the name is arbitrary). The `ids=` parameter is where primary sampling units are declared, as well as any clustering information as a formula ie `~PSU+cluster2id...`. When PSU information is unavailable `ids` is given the value 1 or 0. A `strata=` and `fpc=` are available in order to declare the sampling strata and the variable used for finite population correction. None of these are available in the bsa dataset, and estimation commands will therefore rely on the assumption of simple random sampling.

We can now compute estimates similar estimates as in the previous sections. The code below provides the mean of the left vs right political orientation indicator, as well as its 95% confidence interval:

```
svymean(~leftrigh,bsa.design,na.rm = T)### Computes the mean and its standard error...
```

```
            mean     SE
leftrigh 2.5216 0.0155
```

```
confint(svymean(~leftrigh,bsa.design,na.rm = T)) ### ... and confidence interval
```

```
            2.5 %   97.5 %
leftrigh 2.491277 2.551902
```

And now for the median:

```
svyquantile(~leftrigh,bsa.design,quantiles=.5,na.rm = T)
```

```
$leftrigh
    quantile ci.2.5 ci.97.5         se
0.5      2.4    2.4     2.6 0.05100208

attr(,"hasci")
[1] TRUE
attr(,"class")
[1] "newsvyquantile"
```

Frequency and contingency tables are computed using `svytable()`, which follows the same syntax as `xtabs()`

```
### A frequency table...
round(100*
        prop.table(
          svytable(~RAgeCat,bsa.design)
        ),1)
```

```
RAgeCat
18-24 25-34 35-44 45-54 55-59 60-64   65+
 11.2  17.2  16.1  17.9   7.9   6.8  22.8
```

```
### And a two-way contingency table:

round(100*
        prop.table(
          svytable(~RAgeCat+Rsex,bsa.design)
          ,1),1)
```

```
       Rsex
RAgeCat Male Female
  18-24 51.1   48.9
  25-34 50.2   49.8
  35-44 49.7   50.3
  45-54 49.3   50.7
  55-59 48.8   51.2
  60-64 49.0   51.0
  65+   45.4   54.6
```

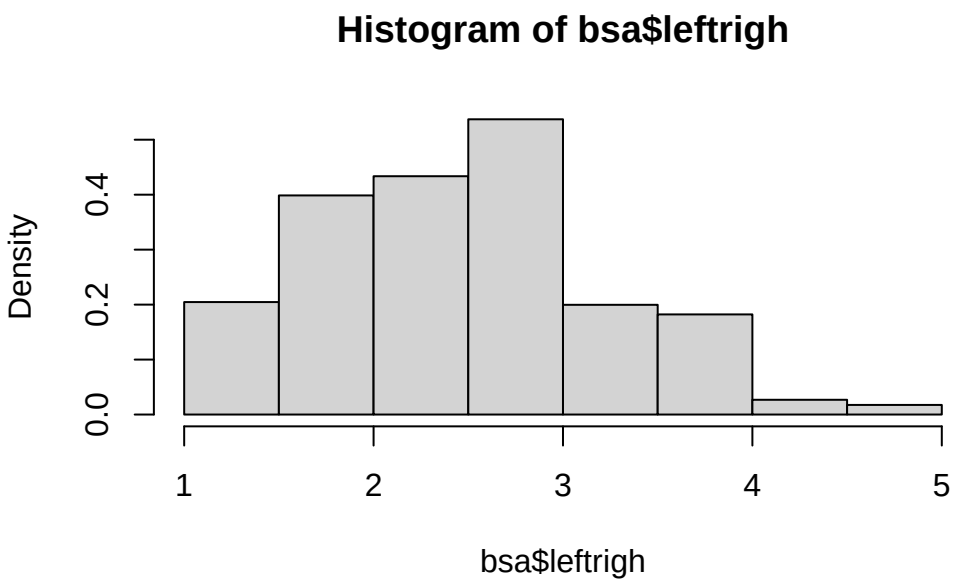
7 Graphs and plots

There are two main ways to produce graphs in R: either by using the straightforward but rather basic plotting commands from the Base package, or the more complex and nicer looking functions from the `ggplot` package.

7.1 Distributional graphs for continuous variables

Graphs such as histograms or box plots are a convenient way to gain a quick overview of the distribution of a variable and are easy to produce. Go back to the BSA data, we can plot the distribution of left-right political orientations scores with the `hist()` command.

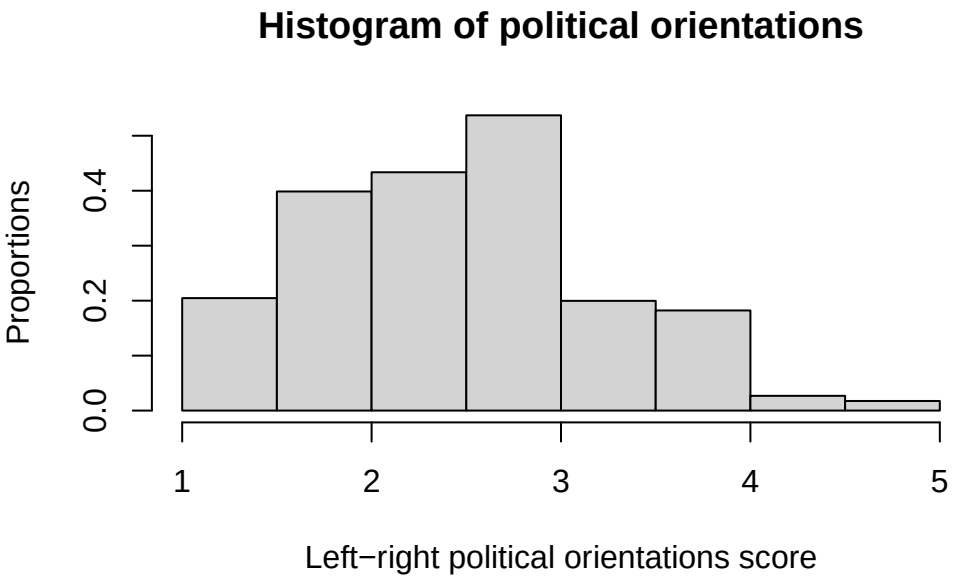
```
hist(bsa$leftrigh,freq=FALSE)
```



The graphs appear visible in the 'Plot' tab on the right hand side of the R Studio window. It shows us that political orientations are slightly skewed towards the left. The `freq=FALSE` option requires the y-axis to be expressed in terms of proportions rather than frequencies.

Titles and labels can easily be added:

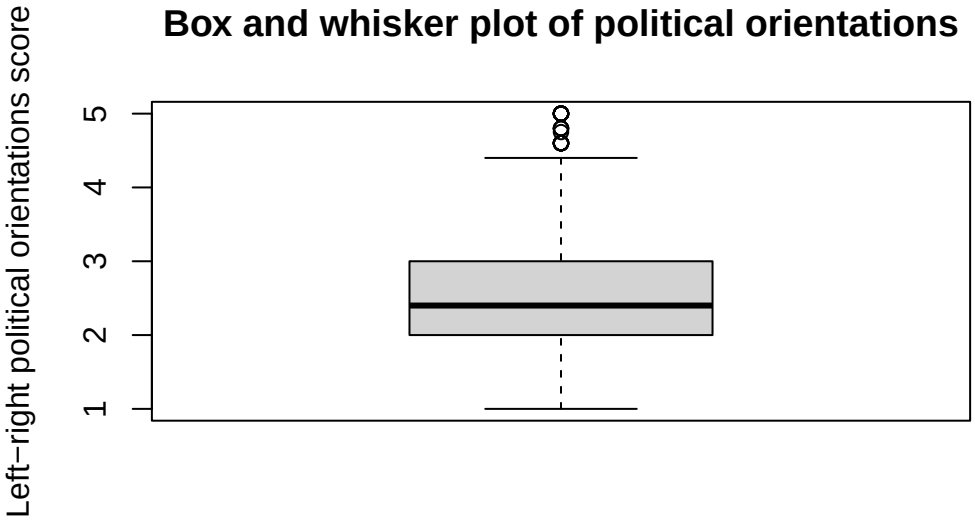
```
hist(bsa$leftrigh,
     freq=FALSE,
     main="Histogram of political orientations",
     ylab="Proportions",
     xlab="Left-right political orientations score")
```



Note that `main`, `ylab` and `xlab` can be used with any Base R plot commands.

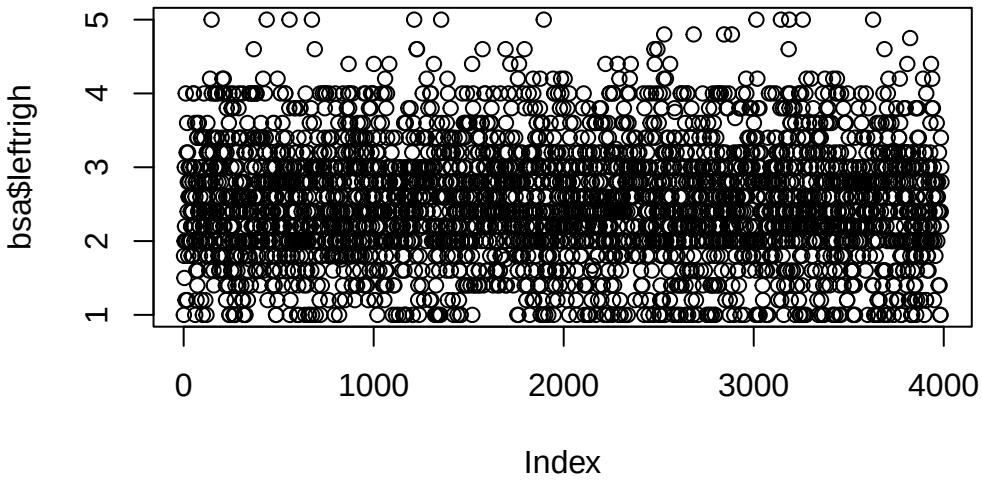
We can also produce a box and whisker plot of the same variable:

```
boxplot(bsa$leftrigh,
        main="Box and whisker plot of political orientations",
        ylab="Left-right political orientations score"
)
```

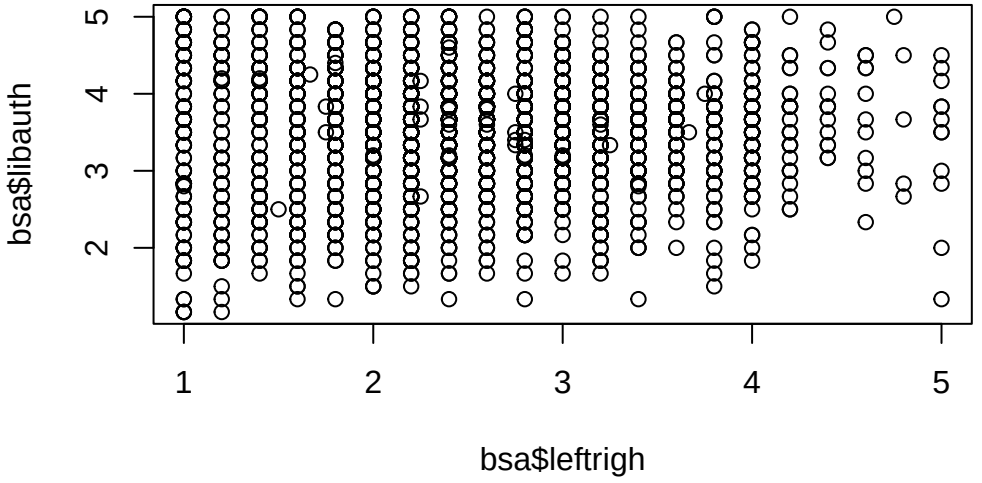


The generic `plot()` command produces one or two-way scatterplots:

```
plot(bsa$leftrigh) ### One way scatter plot of left-right political orientations score
```




```
plot(bsa$leftrigh,bsa$libauth)
```



The second graph shows us that there is little association between the two variables. However, slightly fewer respondents simultaneously score high on the authoritarianism and left vs right scales.

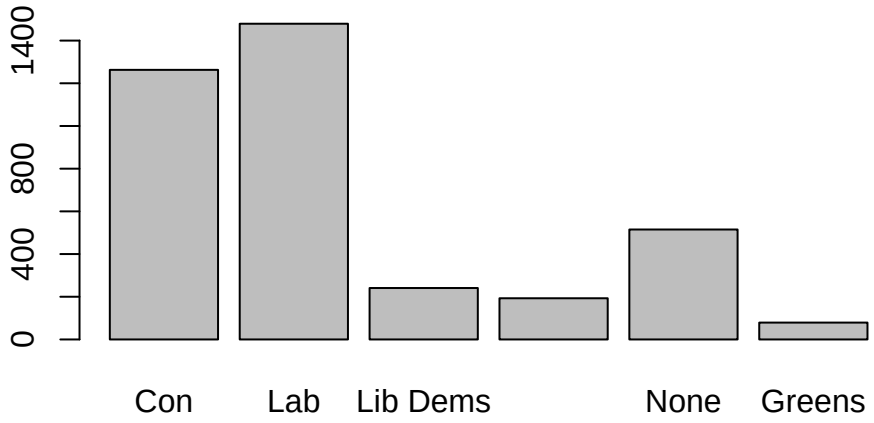
7.2 Plotting categorical variables

The generic `plot()` function provides a quick way to produce bar plots of categorical data. For example, we can examine the distribution of political party affiliations (`Politics` variable) which is a factor (ie categorical) variable. Some preliminary abbreviating of the factor levels are required in order for them to be displayed properly.

```
levels(bsa$PartyId2) ## The third and fourth factor levels are a bit long
```

```
[1] "Conservative"     "Labour"           "Liberal Democrat" "Other party"
[5] "None"             "Green Party"
```

```
levels(bsa$PartyId2)<-c("Con","Lab","Lib Dems","Other","None", "Greens")
plot(bsa$PartyId2)
```



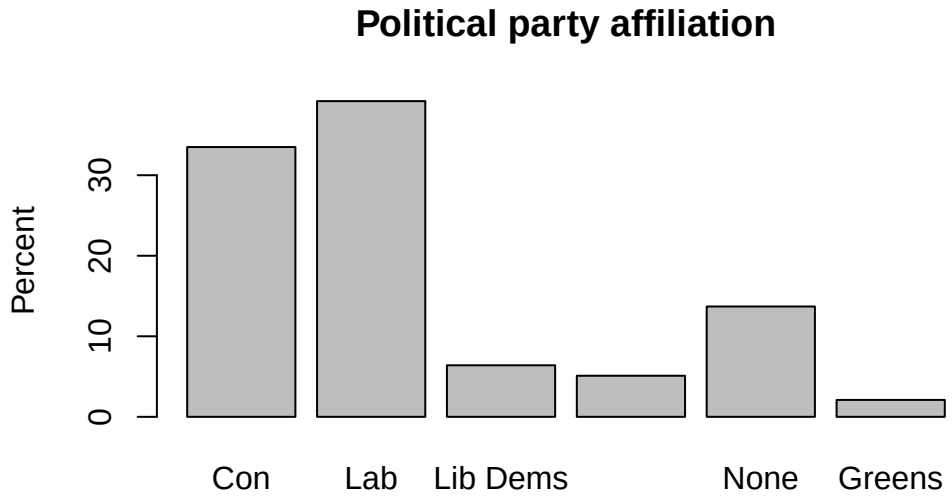
More advanced plots require the barplot() function, which can be used in conjunction with table(). Whereas table() creates the data that will be plotted, barplot() does the actual plotting. For instance, we can produce the same bar plot, but this time with percentages, by creating a frequency table as we did above in Section 5.2, then plot it.

```
party.tab<-round(100*prop.table(
  table(bsa$PartyId2)
),
1)

party.tab
```

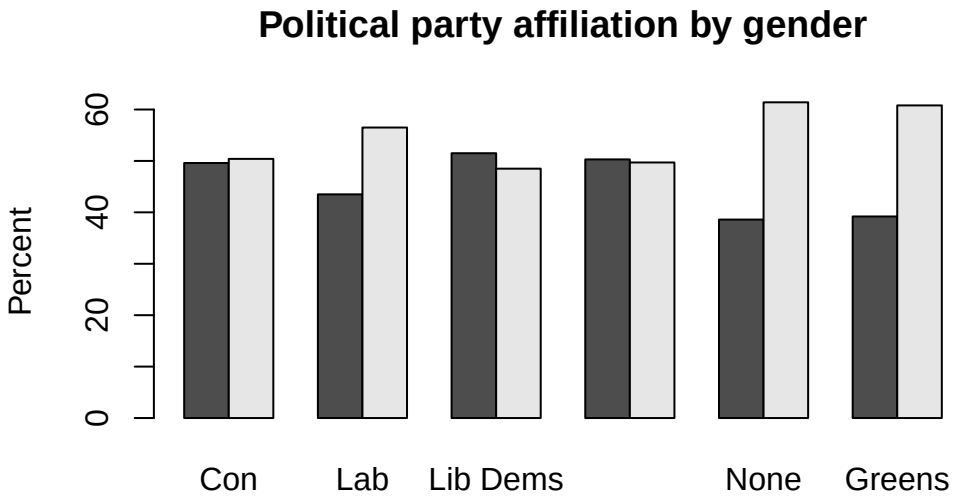
```
     Con      Lab Lib Dems    Other     None   Greens 
    33.5     39.2      6.4      5.1     13.7      2.1 
```

```
barplot(party.tab,
        main="Political party affiliation",
        ylab="Percent")
```



We can go further and create plots for two-way contingency tables of party affiliation by gender. This time we will do it in a single command:

```
barplot(
  round(100*prop.table(
    table(bsa$Rsex,bsa$PartyId2),
    2),              ## Column % (here, gender)
    1),               ## Rounded to 1 decimal
  beside = T,      ## Side-by-side bars
  main="Political party affiliation by gender",
  ylab="Percent")
```



7.3 More advanced plots

Social science research often requires more advanced plots in order to conduct more complex analyses, for instance comparing the mean or median value of a continuous outcome across two or more categorical variables. The `ggplot` package provides one of the most advanced set of tools for plotting data currently available. A few examples are provided below.

Political party affiliation by highest qualification and gender We would like to look at how differences in political party affiliations vary by gender and whether respondents have a degree-level education.

Let us first prepare the data: we need to create the table of result, the proportion of degree vs non degree holders by gender and political party. This is a three-way contingency table, that we can obtain with `ftable()` as shown in Section 5.2, combined with `prop.table()` for the computation of proportions and `round()`. As they are more straightforward to handle in `ggplot`, we convert the table object created by ftable into a data frame. Although we can specify titles and axis labels in the plotting command, it is preferable to keep things simple here and have them already in the the data.
Rather than using the full range of educational achievements recorded in `HEdQual3`, we would like instead to have a dichotomic variable between degree holders and non degree holders. Adding it directly in the ftable command as a boolean expression return a dichotomic variable: “TRUE” for Degree educated respondents, and “FALSE” for everyone else. We just need to

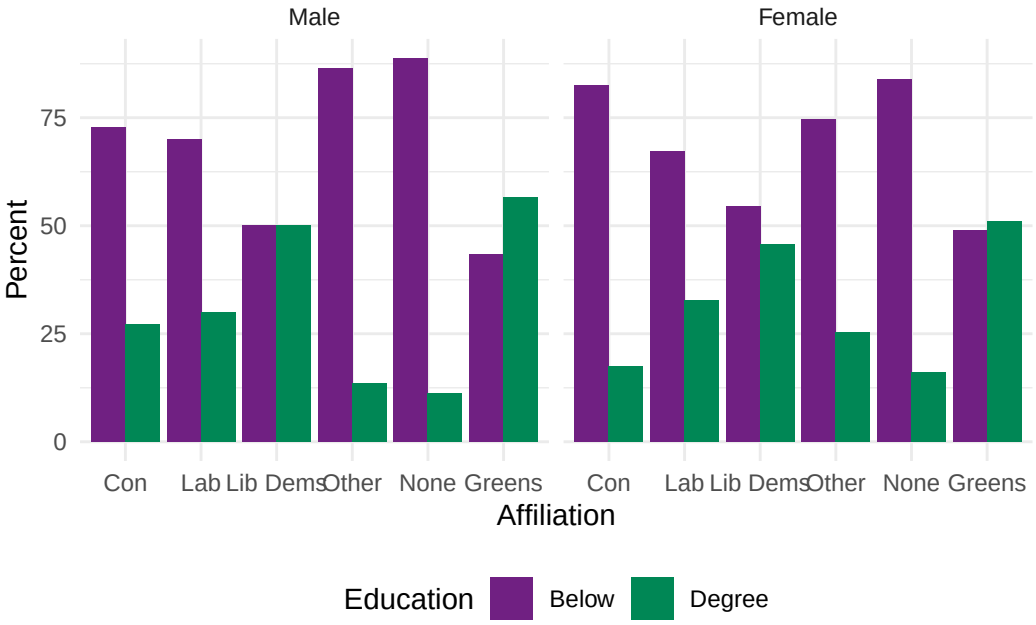
change the levels of this factor variable to make them more intelligible. Finally we change the variable names in our data frame.

```
pa<-round(100*prop.table((ftable(bsa$PartyId2,bsa$Rsex,(bsa$HEdQual3=="Degree"))),1),1)
pa<-data.frame(pa)
levels(pa$Var3)<-c("Below","Degree")
names(pa)<-c("Affiliation","Gender","Education","Percent")
pa
```

```
   Affiliation Gender Education Percent
1          Con   Male     Below    72.7
2          Lab   Male     Below    70.1
3     Lib Dems   Male     Below    50.0
4        Other   Male     Below    86.5
5         None   Male     Below    88.8
6       Greens   Male     Below    43.3
7          Con Female     Below    82.6
8          Lab Female     Below    67.3
9     Lib Dems Female     Below    54.4
10       Other Female     Below    74.7
11        None Female     Below    83.9
12      Greens Female     Below    48.9
13         Con   Male    Degree    27.3
14         Lab   Male    Degree    29.9
15    Lib Dems   Male    Degree    50.0
16       Other   Male    Degree    13.5
17        None   Male    Degree    11.2
18      Greens   Male    Degree    56.7
19         Con Female    Degree    17.4
20         Lab Female    Degree    32.7
21    Lib Dems Female    Degree    45.6
22       Other Female    Degree    25.3
23        None Female    Degree    16.1
24      Greens Female    Degree    51.1
```

We are now ready to plot the data. the `ggplot()` commands usually works as a succession of layers or options that are added to an initial plot specifications. Each extra layer is added after a `+` sign. In the example below, we specify the data and the aesthetic (ie the main parameters of the plot) with the first command: the x and y variables , and the first grouping variable, education). `geom_bar()` stipulates the bar plot, with the ṕosition="dodge" for the bars to be located side by side (position="stack"would have them on top of each other). Finally, facet_wrap() splits the plot by gender.

```
ggplot(data=pa,aes(y=Percent,x=Affiliation,fill=Education))+
  geom_bar(position="dodge",stat="identity")+
  facet_wrap(~Gender)+
  theme_minimal()+                    ### Theme for visualisation
  scale_fill_manual(values=c("#702082", "#008755"))+ ### Custom colours (optional)
  theme(legend.position = "bottom")
```


Male
Female
Percent
75
50
25
0
Con
Lab
Lib Dems
Other
None
Greens
Affiliation
Education
Below
Degree

8 Statistical testing

This section covers how to implement common statistical tests in R with survey data. A working knowledge of these tests and their theoretical assumptions is assumed.

8.1 Differences between means

Two common ways of conducting statistical testing with means in samples consist in testing whether they are significantly different from 0 (one sample t-test), or whether they differ between two groups (two samples t test). In the latter case, one can further distinguish between independent samples (where means come from different groups), or paired samples (when the same measure is taken at several point in time). Given that it is the most widely used in social science, we will only cover the former here.

Several R packages provide functions for conducting t tests. We will be using `t.test()`, part the `stats` package. Suppose we would like to test whether `libauth` significantly differs between men and women. A two sided test is the default (with H_0 that there is no differences between groups), and H_1 that the group means do indeed differ. The test is specified with a formula with on the left hand side the quantity to be tested and on the right hand side the grouping variable.

One sided tests can be conducted by specifying that the alternative hypothesis (H_1) is either **greater** or **less**. `t.test()` assumes that by default the variances are unequal. this can be changed with the `var.equal=T` option.

```
# Testing for significant differences in liberal vs authoritarian score
t.test(libauth~Rsex,data=bsa)
```

```
    Welch Two Sample t-test

data:  libauth by Rsex
t = 1.1622, df = 3011.9, p-value = 0.2452
alternative hypothesis: true difference in means between group Male and group Female is not equal to
95 percent confidence interval:
 -0.02035161  0.07959393
sample estimates:
  mean in group Male mean in group Female
            3.527793             3.498172
```

No significant differences (ie the difference in `libauth` between men and women is not significantly different from zero)

```
# Testing for whether men have a lower (ie more left-wing)  score
t.test(leftrigh~Rsex,data=bsa, alternative="less")
```

```
    Welch Two Sample t-test

data:  leftrigh by Rsex
t = -2.0687, df = 2858, p-value = 0.01933
alternative hypothesis: true difference in means between group Male and group Female is less than 0
95 percent confidence interval:
        -Inf -0.01197607
sample estimates:
  mean in group Male mean in group Female
            2.487564             2.546087
```

Men have a significantly lower score on the scale (at the .05 threshold) and are therefore on average leaning more to the left than women.

8.2 Differences in variance

Another common significance test in social science is the **variance test** which consists of testing whether the variances of the same variable across two groups are equal. This is usually achieved by testing whether the ratio of the variance between the two groups is significantly different from zero. With the BSA data, this amounts to testing whether men and women are more homogenous with regard to their political views.

The syntax for the variance test `var.test()` also included in `stats` is almost identical to that of `t.test()`

```
# Testing for gender differences in liberal vs authoritarian score
var.test(libauth~Rsex,data=bsa)
```

```
    F test to compare two variances

data:  libauth by Rsex
F = 1.0892, num df = 1434, denom df = 1777, p-value = 0.0879
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.9873927 1.2022204
sample estimates:
ratio of variances
          1.089239
```

Significant differences in the variance between men and women, but only at the .1 threshold.

```r
# Testing for whether men have a lower (ie more left-wing)  score
var.test(leftrigh~Rsex,data=bsa,alternative="greater")
```

```
    F test to compare two variances

data:  leftrigh by Rsex
F = 1.3218, num df = 1433, denom df = 1771, p-value = 1.263e-08
alternative hypothesis: true ratio of variances is greater than 1
95 percent confidence interval:
 1.217167      Inf
sample estimates:
ratio of variances
            1.3218
```

The variance of left-right political leaning is larger among men than women, in other words there are more divergence between men than between women.

8.3 Significance of measures of association

Between continuous variables

Another type of common statistical test in social science is about examining whether a coefficient of correlation is significantly different from 0 (alternative hypothesis).

```r
cor.test(bsa$leftrigh, bsa$libauth, use='complete.obs')
```

```
    Pearson's product-moment correlation

data:  bsa$leftrigh and bsa$libauth
t = 0.54472, df = 3202, p-value = 0.586
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.02501074  0.04423951
sample estimates:
        cor
0.009625928
```

As we could have suspected the coefficient of correlation between the two scales is so small that it cannot be said to be significantly different from zero.

Between categorical variables

The chi-square test of independence is a very common test of association between categorical variables. It consists in examining whether the association between two variables is likely to be due to chance or not, in other words whether the variability observed in a contingency table is significantly different from what would be expected were it due to chance.

We will be using `chisq.test()`, also from the `stats` package. By contrast with the test of correlation, the `chisq.test()` needs to be applied to contingency tables that have already been computed. Let us go back to an earlier example, and attempt to test whether the gender differences in political affiliations are due to chance or not.

```
chisq.test(xtabs(~PartyId2 +Rsex,bsa))
```

```
    Pearson's Chi-squared test

data:  xtabs(~PartyId2 + Rsex, bsa)
X-squared = 27.191, df = 5, p-value = 5.236e-05
```

As the R output shows, there are highly significant gender differences in political affiliations ($p<.001$).

9 Regression analysis

The `glm()` command from the R base package is used for fitting linear and non-linear models. These include logistic regression, and more generally models falling under the Generalized Linear Model framework.
In this section, we will use it to investigate the association between the level of education `HEdQual3` and `Voted`, whether someone voted or not. Let's first briefly explore the variables using the `class()` and `table()` commands from the previous chapters:

```
class(bsa$Voted)
```

```
[1] "factor"
```

```
table(bsa$Voted)
```

```
 Yes   No 
2205  776 
```

and

```
class(bsa$HEdQual3)
```

```
[1] "factor"
```

```
table(bsa$HEdQual3)
```

```
                        Degree Higher educ below degree/A level 
                          1050                             1086 
           O level or equiv/CSE                  No qualification 
                          1026                              747 
```

It is a good idea to make sure that categorical variables are stored as factors. This is not absolutely necessary, but gives greater flexibility, for instance when having to change the reference category on the go.

For greater readability we can also shorten the levels of `HEdQual3` using the `level()` function:

```
levels(bsa$HEdQual3) ### The original level names  are a bit verbose...
```

```
[1] "Degree"                          "Higher educ below degree/A level"
[3] "O level or equiv/CSE"            "No qualification"
```

```
### ... Fortunately we can shorten them easily
levels(bsa$HEdQual3) <- c("Degree","A level and above","GCSE or equiv","No Qual")

table(bsa$HEdQual3)
```

```
          Degree A level and above     GCSE or equiv           No Qual
                1050              1086              1026               747
```

What about the levels for our dependent variable? By default, the first level of a factor will be used as the reference category. This can be also checked with the `contrasts()`. In this case, 1 is associated with 'No', so the model will be predicting the probability of NOT voting. If the 1 was associated with 'Yes' then the model will be predicting the probability of voting.

```
levels(bsa$Voted) #Note that Yes comes before No
```

```
[1] "Yes" "No"
```

```
contrasts(bsa$Voted)
```

```
    No
Yes  0
No   1
```

As we are interested in the latter we need to change the reference category using the `relevel()` function. We can also create a new variable named `Voted2` so as to keep a copy of the original variable intact.

```
# Reverse the order
bsa$Voted2 <- relevel(bsa$Voted, ref = "No")

#Check the contrasts
contrasts(bsa$Voted2)
```

```
    Yes
No    0
Yes   1
```

Since the outcome variable (Voted or Voted2) has a binomial distribution, we need to specify to the glm() function that we will be fitting a logistic regression model. We will do this by setting the argument 'family' to 'binomial' and the link function to 'logit'. We could also have used 'probit' instead as a link function. The code below runs the model and stores the result into an object called `fit1`:

```
fit1 <- glm(Voted2 ~ HEdQual3, data=bsa, family=binomial(link=logit))
summary(fit1)
```

```
Call:
glm(formula = Voted2 ~ HEdQual3, family = binomial(link = logit),
    data = bsa)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)                1.49561    0.09188  16.278  < 2e-16 ***
HEdQual3A level and above -0.21342    0.12514  -1.706   0.0881 .
HEdQual3GCSE or equiv     -0.64062    0.12191  -5.255 1.48e-07 ***
HEdQual3No Qual           -0.83672    0.12769  -6.553 5.65e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3297.6  on 2916  degrees of freedom
Residual deviance: 3240.4  on 2913  degrees of freedom
  (1071 observations deleted due to missingness)
AIC: 3248.4

Number of Fisher Scoring iterations: 4
```

To run a model controlling for gender 'Rsex' and age 'RAgeCat', one simply needs to add them on the right hand side of the formula, separated with a plus (+) sign.

```
fit2 <- glm(Voted2 ~ HEdQual3 + Rsex + RAgeCat, data=bsa, family=binomial(link=logit))
summary(fit2)
```

```
Call:
glm(formula = Voted2 ~ HEdQual3 + Rsex + RAgeCat, family = binomial(link = logit),
    data = bsa)

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)                1.11251    0.20044   5.550 2.85e-08 ***
HEdQual3A level and above -0.38676    0.13215  -2.927 0.003427 **
HEdQual3GCSE or equiv     -0.99023    0.13109  -7.554 4.23e-14 ***
HEdQual3No Qual           -1.90625    0.15687 -12.152  < 2e-16 ***
RsexFemale                -0.15708    0.09218  -1.704 0.088363 .
RAgeCat25-34              -0.24604    0.19670  -1.251 0.210996
RAgeCat35-44               0.20668    0.19808   1.043 0.296764
RAgeCat45-54               0.85685    0.20000   4.284 1.83e-05 ***
RAgeCat55-59               0.84062    0.23225   3.619 0.000295 ***
RAgeCat60-64               1.60276    0.25272   6.342 2.27e-10 ***
RAgeCat65+                 2.16408    0.21450  10.089  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3293.1  on 2912  degrees of freedom
Residual deviance: 2922.5  on 2902  degrees of freedom
  (1075 observations deleted due to missingness)
AIC: 2944.5

Number of Fisher Scoring iterations: 4
```

Model interpretation

`Summary()` provide a broad overview of the model output, not dissimilar to other statistical software. We can also examine the content of fit1 and fit2 more in detail and requests a specific element, for example:

```
ls(fit1)
```

```
 [1] "aic"               "boundary"          "call"
 [4] "coefficients"      "contrasts"         "control"
 [7] "converged"         "data"              "deviance"
[10] "df.null"           "df.residual"       "effects"
[13] "family"            "fitted.values"     "formula"
[16] "iter"              "linear.predictors" "method"
[19] "model"             "na.action"         "null.deviance"
[22] "offset"            "prior.weights"     "qr"
[25] "R"                 "rank"              "residuals"
[28] "terms"             "weights"           "xlevels"
[31] "y"
```

```
round(fit1$coefficients,2)
```

```
             (Intercept) HEdQual3A level and above     HEdQual3GCSE or equiv
                    1.50                     -0.21                     -0.64
         HEdQual3No Qual
                   -0.84
```

```
### The coef() function will give the same output:
round(coef(fit1),2)
```

```
             (Intercept) HEdQual3A level and above     HEdQual3GCSE or equiv
                    1.50                     -0.21                     -0.64
         HEdQual3No Qual
                   -0.84
```

It is beyond the remit of this guide to describe the full output of `glm()`. Please refer to the package documentation for more detailed explanations.

Raw logistic regression coefficients measure the effect of variables on the probability of the outcome such as log(betaX)=P(y). It is common practice to convert these into odd ratios by exponentiating them, such as that betaX=exp(P(y)). The following code does this in R:

```
cbind(
  exp(coef(fit2)),exp(confint(fit2))
)
```

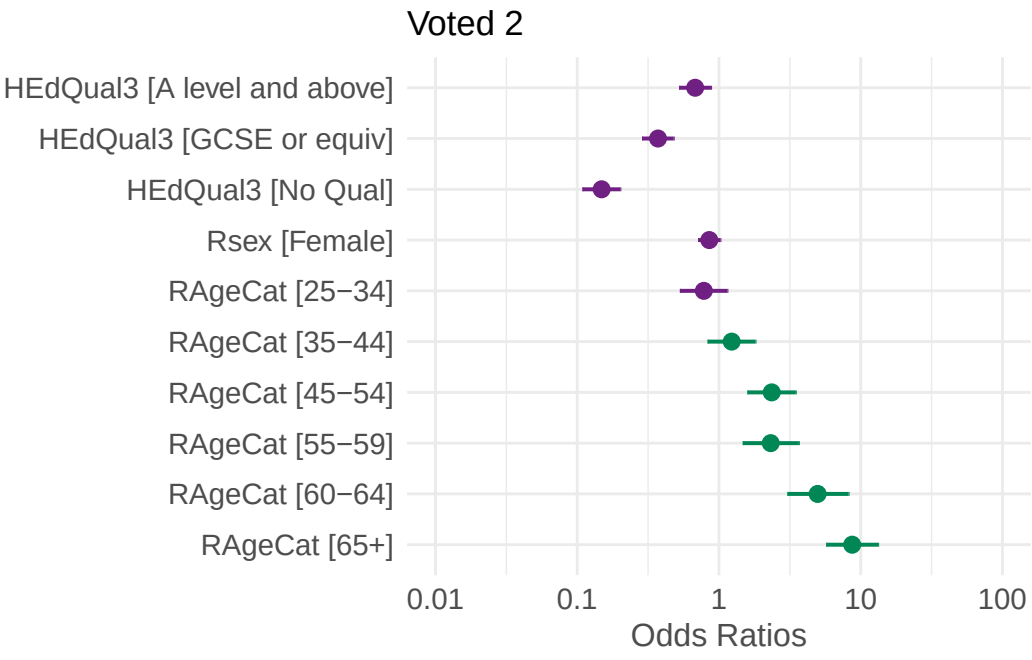
```
                                       2.5 %     97.5 %
(Intercept)                3.0419750 2.0618311  4.5276614
HEdQual3A level and above  0.6792550 0.5237628  0.8795335
HEdQual3GCSE or equiv      0.3714919 0.2868078  0.4796010
HEdQual3No Qual            0.1486366 0.1089442  0.2015607
RsexFemale                 0.8546343 0.7130680  1.0235515
RAgeCat25-34               0.7818917 0.5300310  1.1469983
RAgeCat35-44               1.2295882 0.8316608  1.8095457
RAgeCat45-54               2.3557310 1.5886305  3.4827615
RAgeCat55-59               2.3178122 1.4718049  3.6621685
RAgeCat60-64               4.9667132 3.0428167  8.2090111
RAgeCat65+                 8.7065916 5.7183039 13.2696921
```

Using the `coef()` and `confint()` functions, the code above respectively extracts the coefficients and associated 95% confidence intervals from fit2 then collate them using `cbind()`.

** Plotting the coefficients ** –> We can visualise the odd ratios and their confidence intervals using the `plot.model()` function from the 'sjPlot' package. The 'sjPlot' package needs to be installed and loaded

```
install.packages('sjPlot')
```

```
library(sjPlot)
set_theme(base = theme_minimal()) ### Default sets of options
plot_model(fit2,
           colors = c("#702082", "#008755") ### Added for better accessibility
)
```



Assessing model fit The Akaike Information Criterion (AIC) is a measure of relative fit for maximum likelihood fitted models. It is used to compare the improvement in how several models fit some data relative to each other, allowing for the different number of parameters or degrees of freedom. The smaller the AIC, the better the fit. In order for the comparison to be valid, we need to ensure that the models were run with the same number of observations each time. As it is likely that the second model was run on a smaller sample, due to missing values for the Age and Sex variables, we will need to re-run the first one without.

```
fit1 <- glm(Voted2 ~ HEdQual3, data=bsa%>%
              filter(!is.na(Rsex) & !is.na(RAgeCat)), family=binomial(link=logit))

c(fit1$aic,fit2$aic)
```

```
[1] 3244.507 2944.535
```

We can see that the model controlling for gender and sex is a better fit to the data than the one without controls as it has an AIC of 2944.5 against 3244.5 for fit1.

With the information about the deviance from fit1 and fit2, we can also compute the overall significance of the model, that is whether the difference between the deviance (another likelihood-based measure of fit) for the fitted model is significantly different from that of the empty or null model. This is usually carried by conducting a chi square test, accounting for the differences in the number of parameters (ie degrees of freedom) between the two models. As with other R code, this can achieved step by step or in one go:

```
dev.d<-fit2$null.deviance - fit2$deviance
df.d<-fit2$df.null - fit2$df.residual
p<-1 - pchisq(dev.d, df.d)
c(dev.d,df.d,p)
```

```
[1] 370.5486  10.0000   0.0000
```

The Chi square test indicates that the difference in deviance of 370.5 with 10 degrees of freedom is highly significant (P<.001)

10 Further information

10.1 Additional commands of interest

The following non exhaustive list provides a few examples of commands and packages that tackle common types of analysis which might be relevant to users of large UK surveys:

- Further regression analysis: the `glm()` command can be used for fitting a large number of regression including Poisson and multinomial models. The packages 'lme4' and 'nlme' are used to fit respectively linear and non linear multilevel models, also known as mixed models.
- Complex survey data and analysis commands and functions can be found in the 'survey' package. It includes commands for taking into account stratified and clustered samples, weights compute design effects and confidence intervals, etc..
- For users interested in latent variable modelling the `factanal()` command from the `stats` package conducts factor analysis. Other resources are available in the `poLCA` (Latent Class Analysis), `ltm` (Latent Trait modelling), `sem` (Structural equation modelling) packages. The `Lavaan` package aslso provides a wide range of functions for structural equation modelling including with categorical outcomes.
- Users interested in longitudinal and time series analysis will be interested in the `stats`and the 'tseries' packages. The packages `survival` and `eha` deal with event history and survival analysis, whereas 'grofit' and 'plm' are designed for panel data and growth analyses.

10.2 Additional online resources

There are hundreds of web sites dedicated to R that users can consult, in addition to CRAN and the main R help list, R-Help with its searchable archives. A few of the most common ones are listed here:

- As with other statistical packages, the UCLA website provides a good starting point for beginners
- The University of North Texas provides useful links to R resources
- The R Bloggers website contains many posts about R - in particular useful introductory information
- Stackexchange is not specific to R but contains many forum-type questions and answers raised by R users
- This website presents useful basic information about graphs in R.
- The Centre for Multilevel modeling at Bristol University has several pages dedicated to R users interested in Multilevel modeling

11 References

- R Core Team. (2017). R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.r-project.org/ RStudio Team. (2016).
- RStudio: Integrated Development for R. Boston, USA: RStudio, Inc. Retrieved from http://www.rstudio.com/
- Tennekes, M. (2017) tmap: Thematic Maps. R package version 1.10. Retrieved from https://cran.r-project.org/package=tmap
- Wickham, H., & Francois, R. (2016). dplyr: A Grammar of Data Manipulation. R package version 0.5.0. Retrieved from https://cran.r-project.org/package=dplyr
- Wickham, H. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2009. Retrieved from https://cran.r-project.org/package=ggplot2