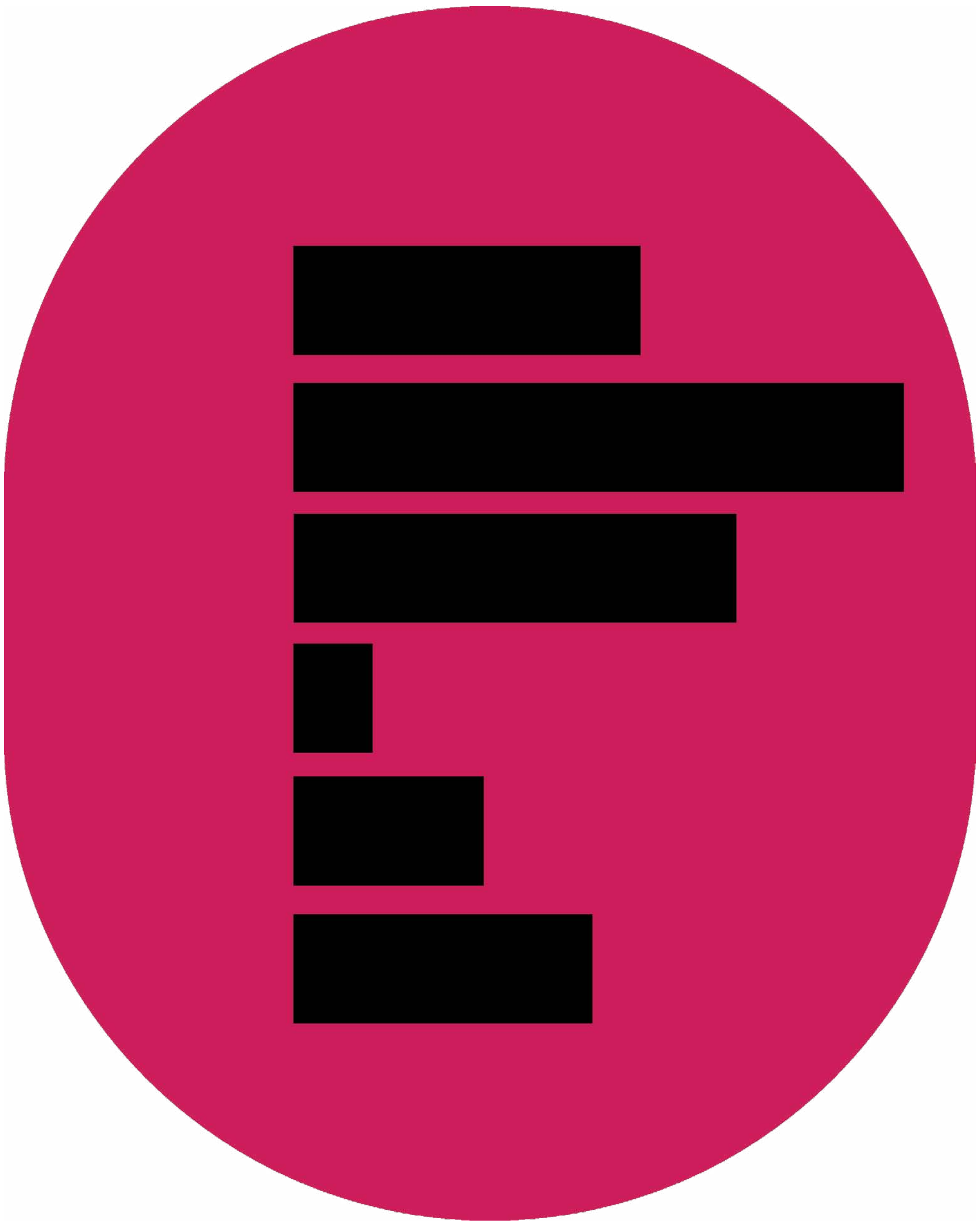
ukdataservice.ac.uk



Using R to analyse key UK surveys

UK Data Service

Author: UK Data Service
Updated: July 2019
Version: 1.3

Contents

1. Introduction....4
  1.1. What is R? 4
  1.2. The pros and the cons of R....5
2. Using R: essential information....7
  2.1. Installing and loading user-written packages....9
  2.2. Getting help 10
  2.3. Interacting with R: command line vs graphical interface....12
  2.4. Objects in R....13
  2.5. Deleting object from the R environment using the rm() function....14
  2.6. Saving in R 14
3. R Studio 17
4. Opening UK Data Service datasets in R 19
  4.1. Which version of UK Data Service datasets can I use in R?....19
  4.2. Variables types....21
5. Essentials of data manipulation.... 23
  5.1. Identifying and selecting variables and observations.... 23
  5.2 Creating and recoding variables.... 24
  5.3. Renaming variables and categories 25
  5.4. Missing values.... 26
6. Descriptive statistics using R - continuous variables.... 29
  6.1 Distribution graphs.... 30
  6.2. Significance testing of correlation 33
  6.3. Tables of summary statistics.... 33
  6.4. Bar charts and plots of summary statistics....35
  6.5 Saving and importing a graph in a Word document.... 38
  6.6 Weighted descriptive statistics.... 39
7. Categorical variables: contingency tables 41
  7.1 One way frequency tables 41
  7.2 Creating bar plots of one-way frequency tables....41
  7.3 Two way or more contingency tables.... 42
  7.4. Test of association between categorical variables 45
  7.5. Univariate and bivariate graphs for categorical variables.... 46
8. Plotting simple maps in R 48
  8.1. Acquiring and downloading the data 48

8.2. Mapping in R – the basic principle 50
8.3 Producing the data needed for mapping 51
8.4 Merging the boundary shapefiles and the data to be plotted 53
8.5. Plotting the map 54
9. Further commands and analyses 56
10. Additional online resources 57
11. References 58

1. Introduction

The aim of this guide is to provide an introduction to analysing large UK surveys with the help of the R statistical software package. This document is targeted at two categories of users:

1. Those outside higher education, or who do not have access to one of the commonly used statistical packages such as Stata, SPSS or SAS (as R is free of charge) 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).

2. More advanced users who are already familiar with one of the aforementioned packages but would like to learn how to carry out their analyses in R. The guide, therefore, focuses on providing step-by-step examples of common operations most users carry out in the course of their research: how to open datasets, do basic data manipulation operations, produce simple descriptive statistics or weighted contingency tables. This is meant to provide the first category of users with a range of procedures that will help them produce straightforward and robust analyses tailored to their needs without spending too much time on learning the inner workings of R. The second category of users will find a number of familiar operations from which they will be able to further expand their R skills.

It should be noted however that this guide is not an introduction to R. Beginners should use it in conjunction with one of the more comprehensive guides available online. Links and information about R resources are available at the end of this document.

Examples provided in this guide, use the Quarterly Labour Force Survey, January - March, 2016, which can be downloaded from the UK Data Service website. The website also has instructions on how to acquire and download large-scale government datasets.

1.1. What is R?

R is a free, user developed, advanced statistical and computing programme. It has a large audience in the programming and statistical community and is increasingly used in the academic world for teaching purposes. 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 same website.

R is often described as an object-oriented statistical programming language rather than simply a statistical analysis package. It originates in the 'S' and 'S Plus' languages developed during the 1970s and 1980s. Anyone can download and use it without charge, and to some extent contribute to and amend the existing programme itself. It is particularly favoured by users who want to develop their own statistical application or implement the latest advances that are not yet available in commercial packages. The existence of a vast number (more than 3,600 at the time of writing this guide) of user written packages – which bear some resemblance to downloadable ado files in Stata – 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 to be followed.

Although R can perform most of the tasks available in generalist statistical packages such as Stata, SPSS, or SAS, it has a broader potential since it can also be used for mapping or data mining. Being a language also means that there are often several ways to carry out analyses in R, each one with its pros and cons. Publication quality output from R can be obtained easily thanks to its integration with the LaTeX document presentation system, and R graphs can also be imported into MS Word documents.

1.2. The pros and the cons of R

Although R has advantages over other statistical analysis software, it has also a few downsides, both of which are summarised below. Users should be reminded that as an 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 good will.

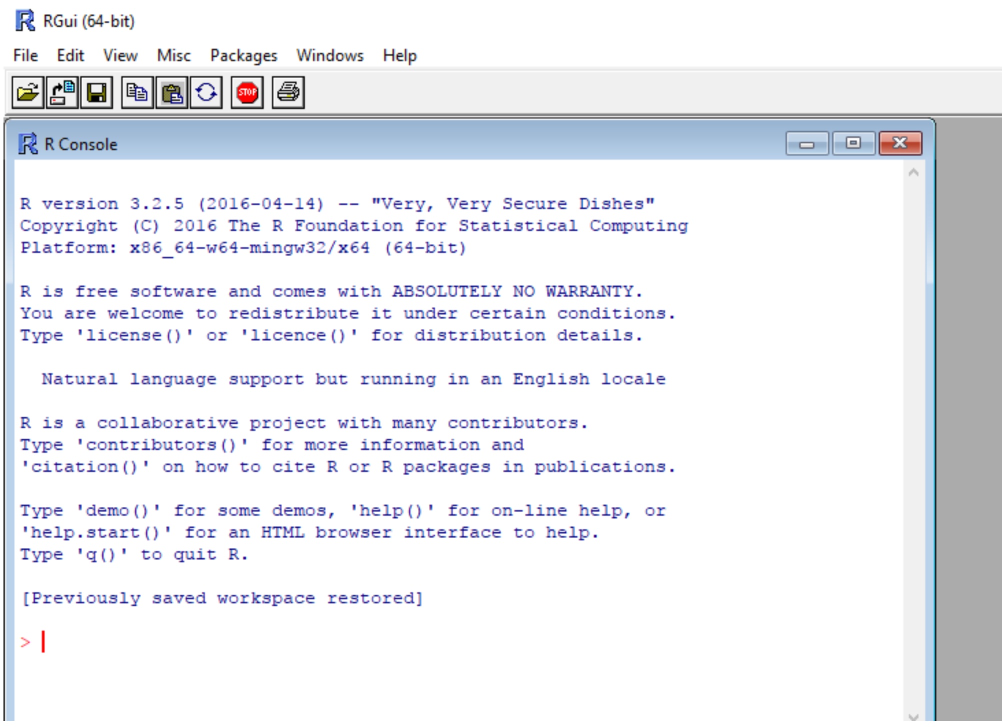
Pros	Cons
R is free, and allows users to perform almost any analysis they want. R puts statistical analysis closer to the reach of individual citizens rather than specialists. Transparency of use and programming of the software and its routines, which improves the peer-reviewing and quality control of the software in many cases Very flexible Availability of a wide range of advanced techniques not provided in mainstream statistical software or only available in specialised packages A very large user base provides abundant documentation, tutorials, and web pages	The learning curve may be steep for users who do not have a reasonably robust background in statistics and programming. Problem solving (for both advanced and beginners) in R may be time-consuming, depending on how common the problem encountered is and may lead to more time spent solving technical rather than substantive issues. Packages can stop being maintained without notice, and some of them have a short life span. Many people who design them are or will become busy academics, and at some point will not have the time to maintain them anymore. Others will take over in some, but not necessary in all the cases.

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

2. Using R: essential information

The R installation programme can be downloaded from the CRAN website and run like any other Windows applications. Versions for Mac and Linux are also available.

After installation, the standard R interface that appears when the programme is launched is shown below. As with advanced statistical packages, the preferred way to interact with R is the command line at the bottom of the R Console and/or by typing commands in a script file (Menu File →New Script). All or selected (highlighted) parts of a script file can be run by typing Control-R.



```
R version 3.2.5 (2016-04-14) -- "Very, Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

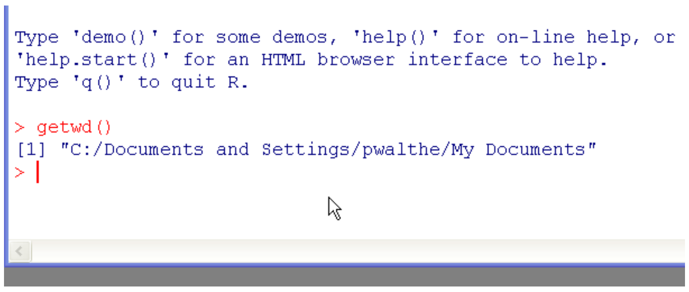
Most R commands adopt the following syntax:

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

Any R command needs to be followed by brackets. In the example shown below getwd is followed by brackets in order for it to understood by R. getwd() identifies which folder R uses by default to store and retrieve files i.e the default working directory

```
>      getwd()
```

The output of the comment is visible below.



```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> getwd()
[1] "C:/Documents and Settings/pwalthe/My Documents"
> 
```

To tell R to use the folder 'R_ESDS' (which needs to be created first) in 'My Documents' as the default location for opening and storing files, one needs to type:

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

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

Notes:

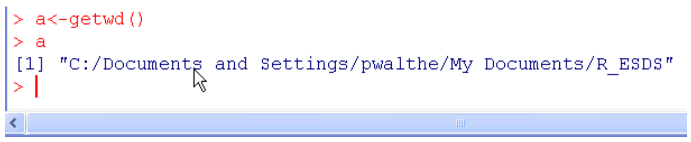
- any character string that is neither a command nor the name of an object (such as a variable name) needs to be put between inverted commas or quotation marks - see the example below about loading user-created packages.
- even when no parameters are specified for a command, brackets are compulsory as shown in the getwd () example above
- R uses forward slashes rather than backslashes (unlike most Windows applications) to separate directories. Using backlashes will return an error message
- although in theory most R commands require 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.

For instance, typing:

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

will store the output of the getwd() command (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:

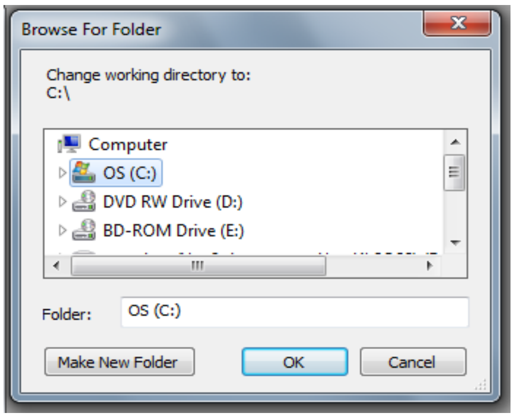


```
> a<-getwd()
> a
[1] "C:/Documents and Settings/pwalthe/My Documents/R_ESDS"
> 
```

The working directory can also be set by using the graphical interface menu:

File... change dir...

A new window will open to select a folder to be used by R as the default directory:



Browse For Folder

Change working directory to:
C:\

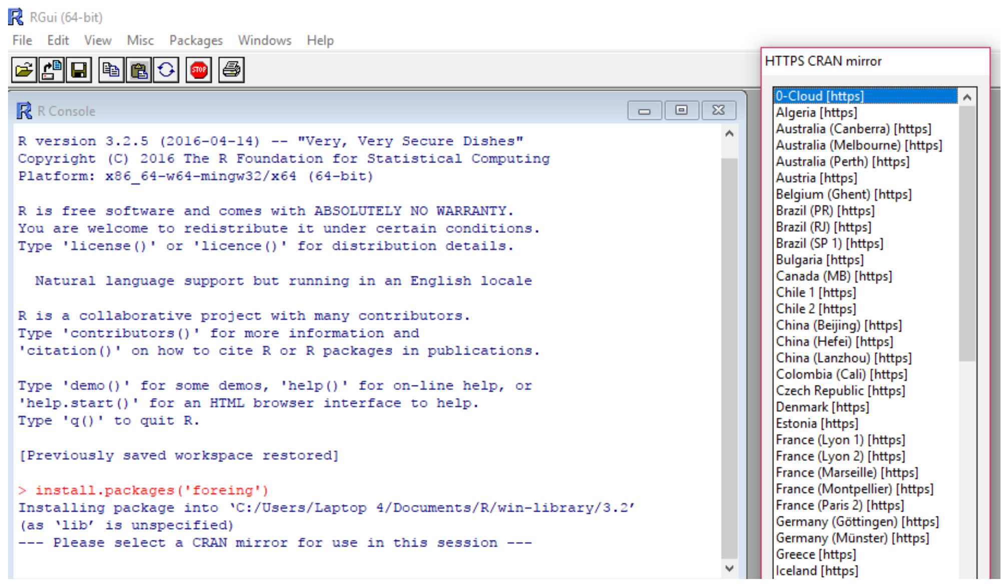
Computer
OS (C:)
DVD RW Drive (D:)
BD-ROM Drive (E:)

Folder: OS (C:)

2.1. Installing and loading user-written 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, in order to install the foreign package which allows users to import Stata or SPSS datasets, one needs to type:

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



```
R version 3.2.5 (2016-04-14) -- "Very, Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> install.packages('foreing')
Installing package into 'C:/Users/Laptop 4/Documents/R/win-library/3.2'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
```

```
0-Cloud [https]
Algeria [https]
Australia (Canberra) [https]
Australia (Melbourne) [https]
Australia (Perth) [https]
Austria [https]
Belgium (Ghent) [https]
Brazil (PR) [https]
Brazil (RJ) [https]
Brazil (SP 1) [https]
Bulgaria [https]
Canada (MB) [https]
Chile 1 [https]
Chile 2 [https]
China (Beijing) [https]
China (Hefei) [https]
China (Lanzhou) [https]
Colombia (Cali) [https]
Czech Republic [https]
Denmark [https]
Estonia [https]
France (Lyon 1) [https]
France (Lyon 2) [https]
France (Marseille) [https]
France (Montpellier) [https]
France (Paris 2) [https]
Germany (Göttingen) [https]
Germany (Münster) [https]
Greece [https]
Iceland [https]
```

A window will appear, prompting user to choose a location where to download the package from. In this case, we can choose for example UK (Bristol). The package is then downloaded and available for use by R. In order to use it however one needs to type

```
>       library(foreign)
```

in order to load it into the memory.

```
>       library()
```

Will tell users which libraries have been downloaded and can be loaded in memory.

For users who feel more comfortable using 'click-and-point', there is also the option to use 'Install Packages' from the Packages tab in the main R window. This will display a list of packages available in alphabetical order for the user to choose from. Next, select the desired package, double click on it and press 'OK' for the installation to begin.

2.2. Getting help

The standard help system in R (unless otherwise chosen at the time of installation) relies on the default web browser (Firefox or Internet Explorer in most cases) to display pages. 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*.

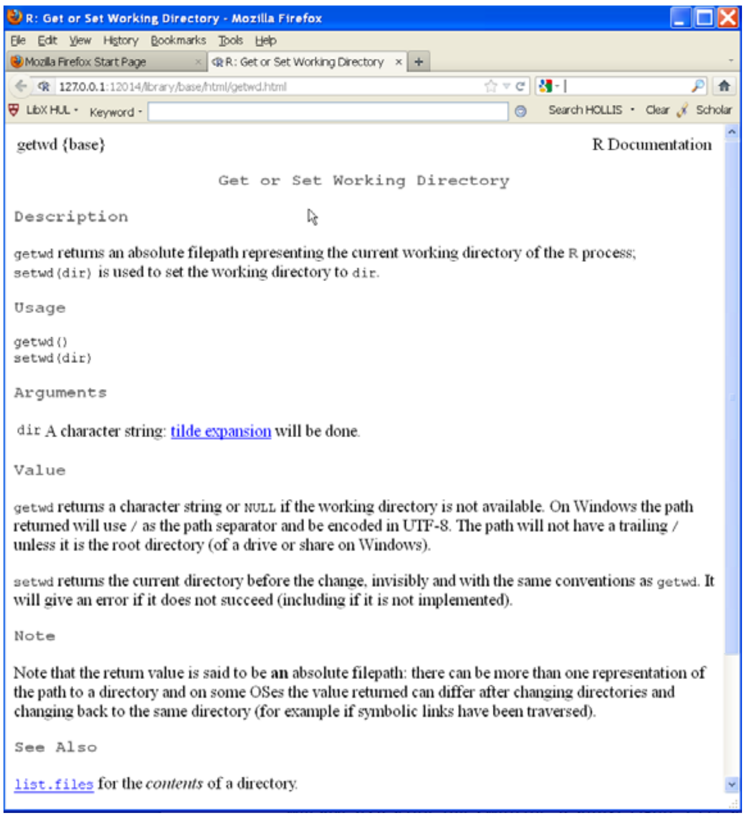
Typing

```
>       ?getwd
```

is the equivalent of:

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

and will open the help page for the getwd() command in the default web browser.


R: Get or Set Working Directory - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Mozilla Firefox Start Page
R: Get or Set Working Directory
127.0.0.1:12014/library/base/html/getwd.html
LibX HUL Keyword
Search HOLLIS Clear Scholar

getwd {base}   R Documentation

Get or Set Working Directory

Description

getwd returns an absolute filepath representing the current working directory of the R process; setwd(dir) is used to set the working directory to dir.

Usage

```
getwd()
setwd(dir)
```

Arguments

dir A character string: tilde expansion will be done.

Value

getwd returns a character string or NULL if the working directory is not available. On Windows the path returned will use / as the path separator and be encoded in UTF-8. The path will not have a trailing / unless it is the root directory (of a drive or share on Windows).

setwd returns the current directory before the change, invisibly and with the same conventions as getwd. It will give an error if it does not succeed (including if it is not implemented).

Note

Note that the return value is said to be **an** absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the same directory (for example if symbolic links have been traversed).

See Also

list.files for the *contents* of a directory.

This will work for any command directly available in memory (ie in the default package or those in the 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 the available documentation for that keyword

```
>       ??foreign
```

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

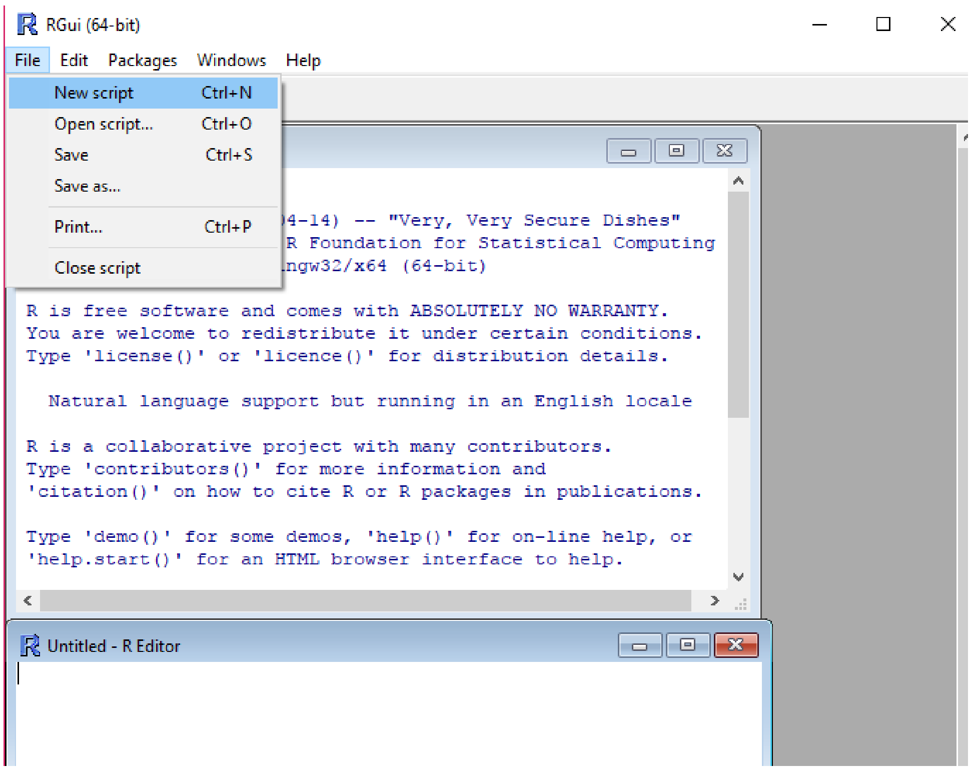
```
>       help(package='foreign')
```

This only work 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.

2.3. Interacting with R: command line vs graphical interface

As with other statistical packages, most users will want to write their programme in a script file, similar to the 'do' file in Stata or syntax file in SPSS. Most users will also save their programmes as R script files (files ending with the .R suffix). To open a script in R, one needs to select 'New script' in the 'File' menu – this will produce a script window in which to type commands:



```
)4-14) -- "Very, Very Secure Dishes"
R Foundation for Statistical Computing
ngw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
```

Within a script window, the whole file or a selection of commands can be run by pressing CTRL-R.

On the other hand, several graphical user interfaces (GUI) have been developed for R users unfamiliar with writing syntax files. RKward, Deducer R Commander are a few examples of such GUIs. Some of these (such as R Commander) target beginners or students and provide access to only a subset of simple statistical analysis tools, while others cover most of the R commands.

The GUI interfaces are different from the R basic interface and have their own learning curve so they will not be described in detail in this document. Below however is a brief overview of how to install and use R Commander.

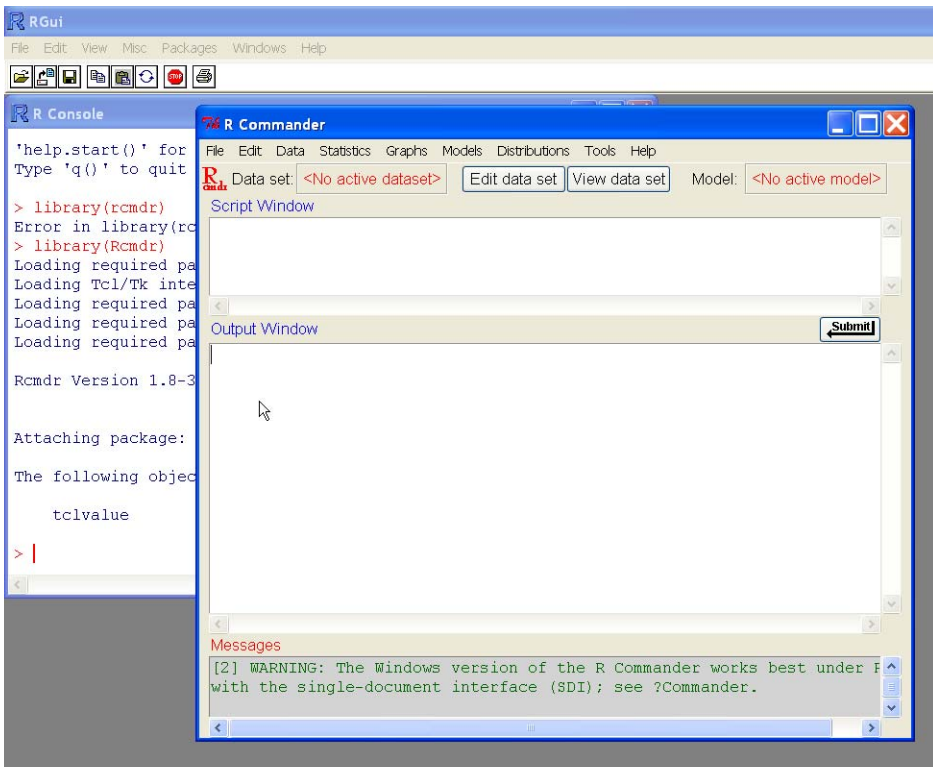
R commander is in fact an R package as any other one. It can be installed by typing:

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

The difference with other R packages is that loading the package in memory will actually launch R Commander:

```
>      library(Rcmdr)
```

Below is a screenshot of R Commander, which shows some of the functions it offers: descriptive statistics, graphs, data editing, generation of random distributions. See the [R Commander documentation] for more information.


RGui
File Edit View Misc Packages Windows Help
R Console
R Commander
File Edit Data Statistics Graphs Models Distributions Tools Help
Data set: <No active dataset>
Edit data set
View data set
Model: <No active model>
Script Window
Output Window
Submit
Messages

```
'help.start()' for
Type 'q()' to quit

> library(rcmdr)
Error in library(rc
> library(Rcmdr)
Loading required pa
Loading Tcl/Tk inte
Loading required pa
Loading required pa
Loading required pa

Rcmdr Version 1.8-3


Attaching package:

The following objec

    tclvalue

>
```

```
[2] WARNING: The Windows version of the R Commander works best under F
with the single-document interface (SDI); see ?Commander.
```

.

2.4. Objects in R

R is an object oriented language, which means that almost any information it uses is stored as 'objects' – ie 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 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 directly available to most users. Common object classes include factors (roughly 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.

2.5. Deleting object from the R environment using the rm() function

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

```
>      rm(x)
>      rm(mydata)
```

This only works with R objects; if we want to delete a specific variable of a dataset, we need a different function. For example, to delete the variable 'age' of the dataset 'mydata', an option is to set the variable to NULL:

```
>      mydata$age<-NULL
```

There are of course other alternatives to delete variables from a dataset. But these are beyond the scope of this introductory guide.

2.6. Saving in R

When working with data, it is very likely that the user will edit the original dataset, either by recoding variables or creating new ones, etc. In those cases, saving the progress made in the data used is crucial to avoid repeating every single operation in the next session working with the data.

There are several ways of accomplish this, depending on the format in which the data will be stored.

The line of command used to save the data frame used called "mydata" is:

```
>     save(mydata, file="mydata.Rda")
```

This command will save the data into a format that can be read by R. The first part of the command is referring to the data frame used in the current R session, while the section file="mydata.Rda", is referring to the data that will be saved in the working directory. The name of the saved file can be changed, for instance:

```
>     save(mydata, file="mydata_Jan17.Rda")
```

To load the saved .Rda data:

```
>     load("mydata_Jan17.Rda")
```

This command will work only if the working directory where the data is stored is defined in advance (see section 2). Alternatively, the path to the folder where the data is saved can be specified.

```
>     load(‘c:/mydocuments/mydata_Jan17.Rda")
```

Another option to save the data is using the “foreign” package, so data can be exported to several formats, such as .txt, .cvs, .dta, which can be used in other software packages.

The following example shows how to export data from R to a comma delimited format (.csv) that can be read in excel, Stata and SPSS.

```
>     library(foreign)

>     write.csv(mydata, “mydata.csv”, row.names=FALSE)
```

Another example is to export the data as a Stata file, using the foreign package previously loaded:

```
>     write.dta(mydata, "mydata.dta")
```

The newly created files will be stored in the working directory defined earlier on.

Some users will want to save the whole R project in which they are working. This would include functions, variables, data (in R it is possible to load and work with more than one dataset at a time). This option comes very handy, especially when working with several datasets.

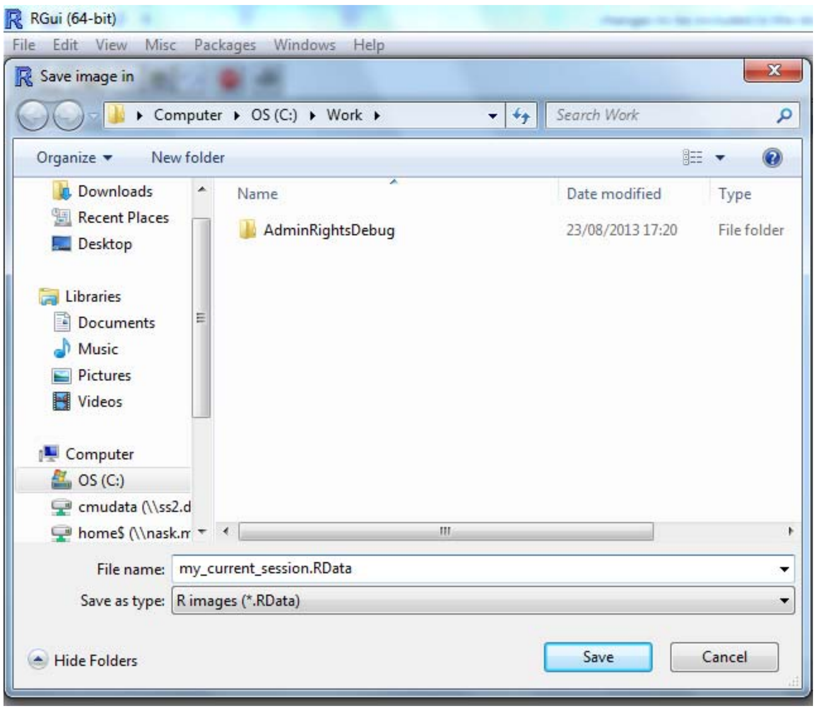
Thus, another approach is to save the current session or workspace as an image, using the *save.image* command, specifying the path where the workspace will be saved. For example, to save ‘my current session’ in R, one needs to write the following command, making sure to include the ***.RData*** extension.

```
>     save.image("c:/Folder/my_current_session.RData")
```

Instead, a workspace can also be saved using the graphical interface:

File... Save workspace...

This will open a window to save the workspace in a particular folder, which can be different from the current working directory. The name needs to be specified in the 'File name' section. The workspace will be saved in the chosen folder as an .RData file.



File name: my_current_session.RData

Save as type: R images (*.RData)

The data can be retrieved using the load function

```
>     load("c:/Folder/my_current_session.RData")
```

Or using graphical interface as follows:

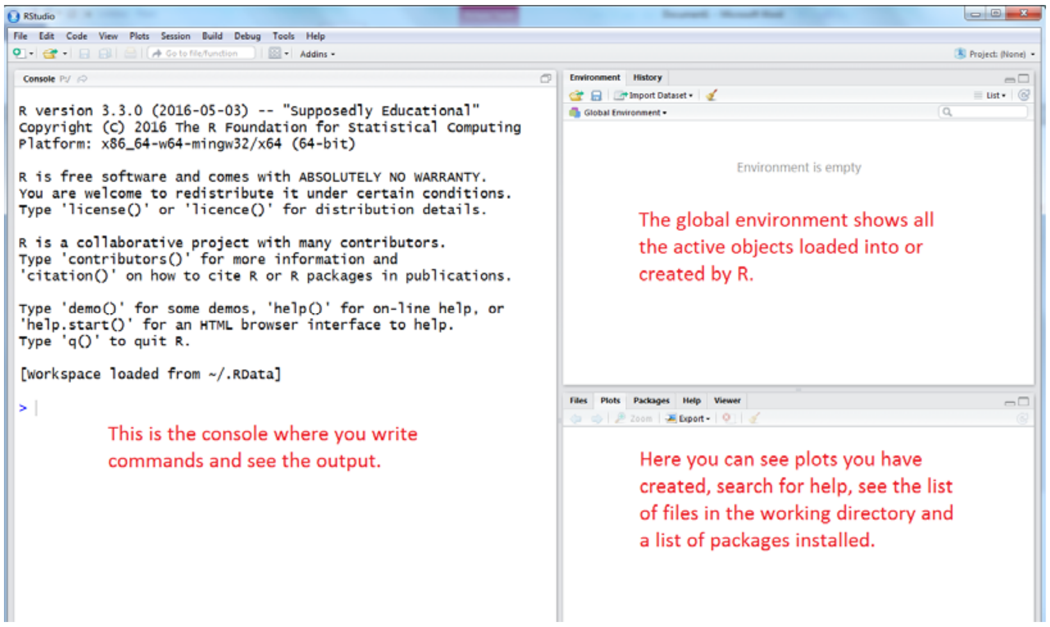
File... load workspace...

3. R Studio

There is a more user-friendly way of using R that is becoming increasingly popular among R users, which is R Studio. This software package is available to download from this website: https://www.rstudio.com/.

In R Studio, R syntax works exactly the same as with the traditional R environment, but the interface is more interactive, which makes it easier to use.

This is the main screen of R Studio:



```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[workspace loaded from ~/.RData]

>
```

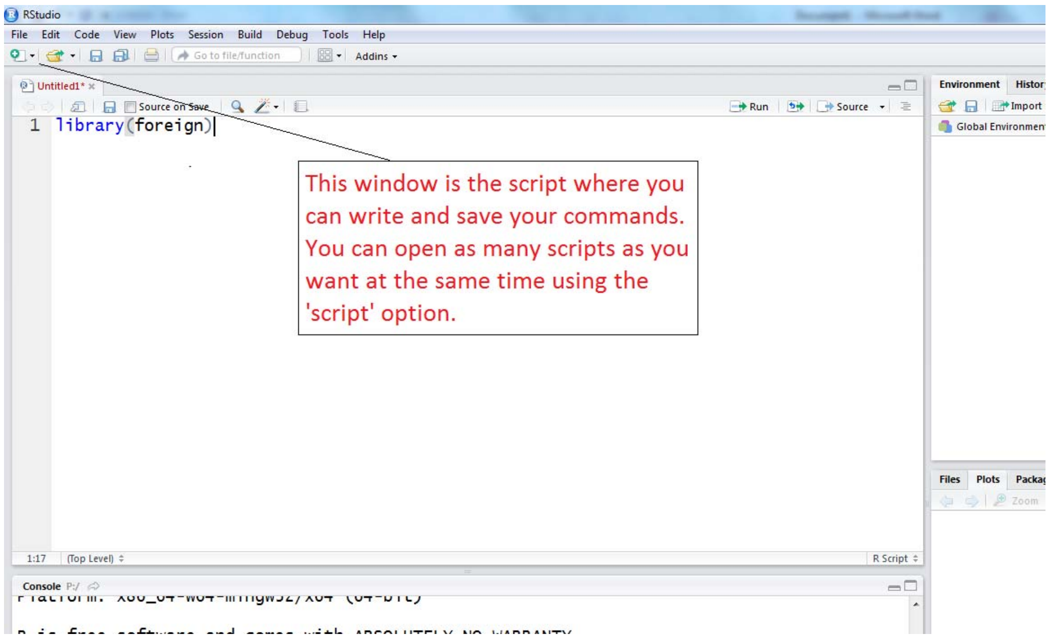
RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Project: (None)
Console
Environment History
Import Dataset
List
Global Environment
Environment is empty
The global environment shows all the active objects loaded into or created by R.
This is the console where you write commands and see the output.
Files Plots Packages Help Viewer
Zoom Export
Here you can see plots you have created, search for help, see the list of files in the working directory and a list of packages installed.

To create a new script, go to:

File… New File… New script…

Or just press the 'Plus' green button, as shown in the image below.

You can either select the command or just place the cursor on it, then press 'Run' or type 'Ctrl+R'



```
library(foreign)
```

This window is the script where you can write and save your commands. You can open as many scripts as you want at the same time using the 'script' option.

The basics of how to use R have already been revised in the previous sections. In this section, we only work in an example of how to use R, via R Studio.

4. Opening UK Data Service datasets in R

In this guide, we use the Quarterly Labour Force Survey, January - March, 2016 (LFS), which can be downloaded from the UK Data Service website. The website also has instructions on how to acquire and download large-scale government datasets.

4.1. Which version of UK Data Service datasets can I use in R?

With the commands read.dta() and read.spss(), R can open Stata and SPSS datasets. In this guide, we will work with the Stata version of the LFS data.

First, if we haven't done it already, we need to load the package 'foreign' into the memory by typing the following command on the script screen and clicking 'Run' (or Ctrl+R); the cursor needs to be on the same line of the command. Alternatively, the command can be selected before clicking 'Run'.

```
>      library(foreign)
```

We then set the default working directory

```
>      setwd("C:/Documents        and        Settings/user/My
Documents/R_ESDS/UKDA-7985-stata11/stata11")
```

This way, we won't have to specify the full path of files that we are will be opening or saving. We can finally open the file:

```
>      lfs<-read.dta("lfsp_jm16_eul.dta")
```

Note: This may take some time to complete depending on the size of the dataset. R will issue warnings, which can be safely ignored. These have to do with the way Stata value labels are imported and does not affect data analysis.

In R jargon, we have loaded the content of the LFS file into an R object which is called a 'data frame'. By contrast with Stata and SPSS, R allows users to open several datasets simultaneously, so we could for example load another issue of the LFS into the memory. The number of datasets that may be opened simultaneously is only limited by the computer's physical memory. R uses by default all the memory available on your computer if necessary. This means however that the programme can become slow if you open a very large dataset on a computer with a small amount of physical memory (<1GB on latest versions of Windows)

Note: The foreign package can only read Stata files up to version 12[1]. To import data from Stata version 13 and above, the data need to be saved in Stata 12 version. The Stata command saveold is useful for this.

To save the file 'example_v13.dta' (already loaded in Stata) in Stata 12 version, use the following command:

[1] Package 'foreign'.https://cran.r-project.org/web/packages/foreign/foreign.pdf

```
saveold example_v13, version(12)
```

The data will be saved in a version compatible with the ‘foreign’ package of R, so it can be imported into R using the ‘read.dta’ function.

There are other alternatives to import data from Stata (up to version 14), SAS and SPSS. This is the newly created package called ‘haven’[2] by Hadley Wickham and Evan Miller.

We can find the number of observation and variables in the dataset by typing

```
>	dim(lfs)
```

One can see that there are 90787 observations and 768 variables in the dataset

Typing:

```
>	ls()
```

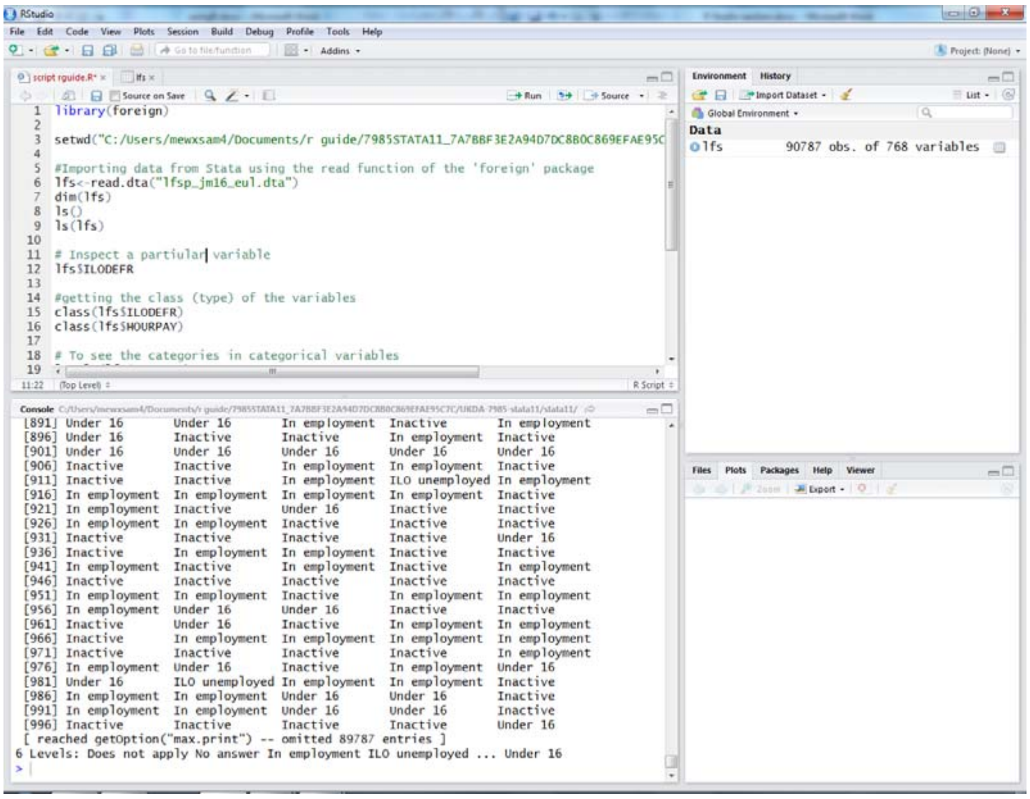
will show us that the object ‘lfs’ has appeared, but what if we want to get the list of all variables in the dataset? We will need to type

```
>	ls(lfs)
```

Now, what if we want to inspect a particular variable, for example the ILO employment status of respondents (this is the variable called ILODEFR in the LFS)? We would need to type:

```
>	lfs$ILODEFR
```

[2] https://cran.r-project.org/web/packages/haven/haven.pdf



```
library(foreign)

setwd("C:/Users/mewxsam4/Documents/r guide/7985STATA11_7A7BBF3E2A94D7DC8B0C869EFAE95C

#Importing data from Stata using the read function of the 'foreign' package
lfs<-read.dta("lfsp_jm16_eul.dta")
dim(lfs)
ls()
ls(lfs)

# Inspect a partiular variable
lfs$ILODEFR

#getting the class (type) of the variables
class(lfs$ILODEFR)
class(lfs$HOURPAY)

# To see the categories in categorical variables
```

```
Data
lfs          90787 obs. of 768 variables
```

```
[891] Under 16      Under 16      In employment Inactive      In employment
[896] Under 16      Inactive      Inactive      In employment Inactive
[901] Under 16      Under 16      Under 16      Under 16      Under 16
[906] Inactive      Inactive      In employment In employment Inactive
[911] Inactive      Inactive      In employment ILO unemployed In employment
[916] In employment In employment In employment In employment Inactive
[921] In employment Inactive      Under 16      Inactive      Inactive
[926] In employment In employment Inactive      Inactive      Inactive
[931] Inactive      Inactive      Inactive      Inactive      Under 16
[936] Inactive      In employment In employment Inactive      Inactive
[941] In employment Inactive      In employment Inactive      In employment
[946] Inactive      Inactive      Inactive      Inactive      Inactive
[951] In employment In employment Inactive      In employment In employment
[956] In employment Under 16      Under 16      Inactive      Inactive
[961] Inactive      Under 16      Inactive      In employment In employment
[966] Inactive      In employment In employment In employment In employment
[971] Inactive      Inactive      Inactive      Inactive      In employment
[976] In employment Under 16      Inactive      In employment Under 16
[981] Under 16      ILO unemployed In employment In employment Inactive
[986] In employment In employment Under 16      Under 16      Inactive
[991] In employment In employment Under 16      Under 16      Inactive
[996] Inactive      Inactive      Inactive      Inactive      Under 16
 [ reached getOption("max.print") -- omitted 89787 entries ]
6 Levels: Does not apply No answer In employment ILO unemployed ... Under 16
>
```

Typing the name of a variable in a data frame will list the first 1000 observations for that variable. Other commands provide more useful information, such as summary(). See the 'Variable types' section below for more details.

Since an R session may involve several datasets at the same time, having variable names immediately recognised when typed requires that we 'attach' a dataset as the default data frame.

```
>       attach(lfs)
```

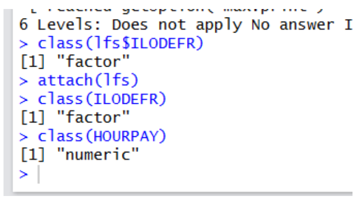
We can now omit the lfs$ prefix when specifying a variable in our analyses.

Despite the practicalities of the ***attach/detach*** command, it is always advised to use this code with caution, since the risk of confusion is greater, especially when working with several datasets at a time and when the same variable names are used in different datasets. Many online references can be found advising about whether to use or not the ***attach*** function. For instance, on the UCLA website , there is a brief explanation of the main issues.

4.2. Variables types

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 object. If one is unsure about which class an object belongs to, one can use the class() command in order to find out.

```
>       class(lfs$ILODEFR)

>       class(ILODEFR)
```

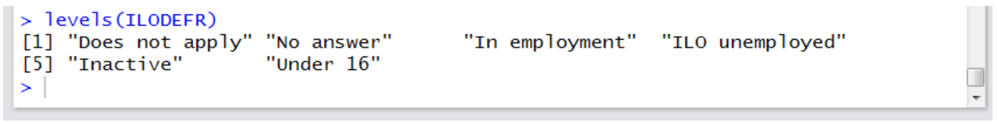


```
6 Levels: Does not apply No answer I
> class(lfs$ILODEFR)
[1] "factor"
> attach(lfs)
> class(ILODEFR)
[1] "factor"
> class(HOURPAY)
[1] "numeric"
>
```

In the case of a categorical variable,

```
>       levels(ILODEFR)
```

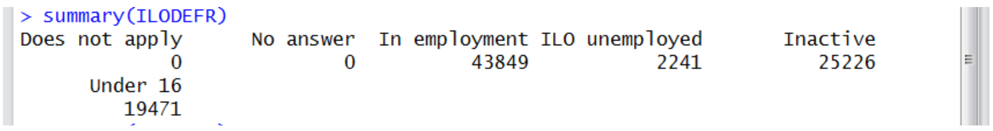
will return the categories of ILODEFR:



```
> levels(ILODEFR)
[1] "Does not apply" "No answer"      "In employment"  "ILO unemployed"
[5] "Inactive"       "Under 16"
>
```

There are several ways to get basic information about a variable in R. One of the most common is the summary() command. A convenient feature of summary() is that it recognizes objects belonging to different classes and treat them accordingly without returning an error message. Typing

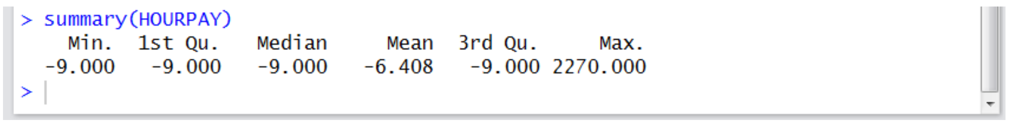
```
>       summary(ILODEFR)
```



```
> summary(ILODEFR)
Does not apply      No answer   In employment ILO unemployed       Inactive
             0              0          43849           2241          25226
      Under 16
         19471
```

will return the frequencies of each category of ILODEFR, whereas:

```
>       summary(HOURPAY)
```



```
> summary(HOURPAY)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  -9.000   -9.000   -9.000   -6.408   -9.000 2270.000
>
```

will compute basic descriptive statistics (mean, median, quartiles, maximum and minimum) in the case of a continuous variable. R will decide automatically which type of summary is best suited to a given object class. Typing summary(lfs) will return summary information about each variable in the LFS dataset.

5. Essentials of data manipulation

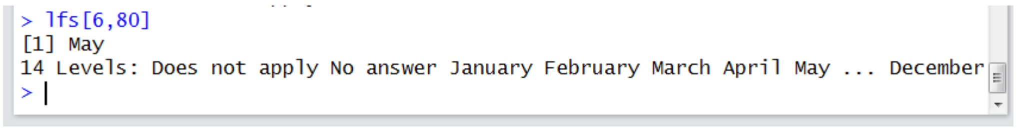
Before attempting to produce statistical analyses in R we need to learn how to recode variables and deal with missing data.

5.1. Identifying and selecting variables and observations

Data frames consist of rows and columns, in the same way as Stata or SPSS datasets. Each row and column can be identified by its number between square brackets: data.frame[row,colums]

For example, if we type:

```
>       lfs[6,80]
```



```
> lfs[6,80]
[1] May
14 Levels: Does not apply No answer January February March April May ... December
> |
```

we will get the value of the sixth observation of the 80th column in the data – 'May' , which happens to be the variable called CONMON (Month started current job). By default, R also provides the names of the levels of CONMON (remember, CONMON, a categorical variable is a *factor* in R whose categories are *levels)*. Therefore, typing

```
>       CONMON[6]
```

will return the same value (ie the first observation of CONMON) – provided we have attached the LFS dataset in the memory using the attach() command.

By contrast, typing

```
>       lfs[,80]
```

would have returned all the first 99,999 values of CONMON, which is equivalent to typing:

```
>       CONMON
```

Following the same logic, we can select observations of a variable based on its values with a conditional statement:

```
>       summary(HOURPAY[HOURPAY>0])
```

will provide basic summary information of HOURPAY only for those with a value greater than 0 (i.e non-missing values), whereas

```
>       summary(HOURPAY[SEX=='Female' & HOURPAY>0])
```

will compute the mean hourly pay for women who have positive earnings, that is who are actually employed since negative values are non responses. Similarly to

Stata, '=' used to evaluate an expression (such as in 'is the value of a variable equal to another?') is represented by '=='

5.2 Creating and recoding variables

There are different ways to recode variables in R. Below we follow a logic that is familiar to users of SPSS and Stata.

Creating a new variable in R is straightforward. For example, we can type:

```
>      lnhourpay <- log(HOURPAY)
```

which will create a new variable called lnhourpay which contains the log value of hourly earnings,

Since the 'lfs' data has been 'attached' using the 'attach' function, the previous command creates a free vector (variable). This means that lnhourpay is not part of our lfs dataset (this is one of the issues related to the use of 'attach'). If we want this new variable to be stored in the 'lfs' data, we would need to write the following command:

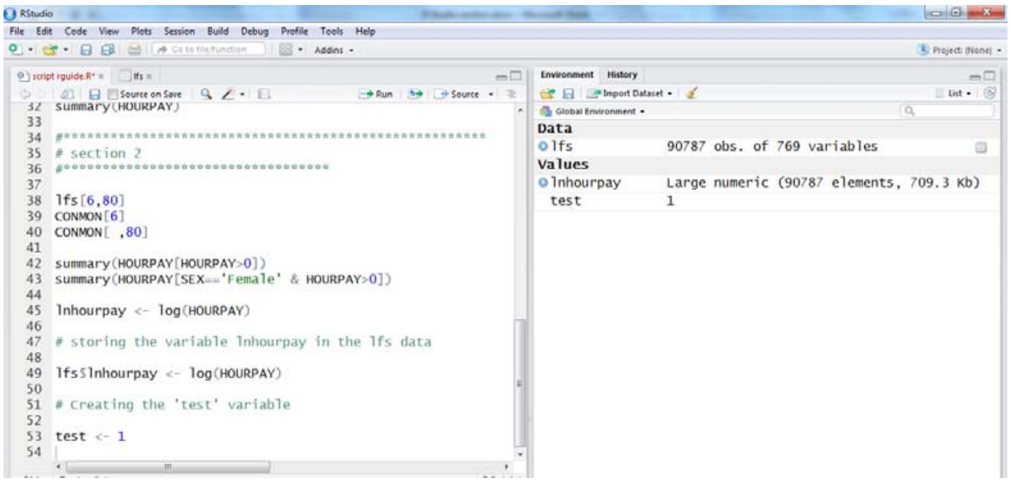
```
>      lfs$lnhourpay <- log(HOURPAY)
```

This command tells R that we are creating the variable 'lnhourpay' and we are storing it in the lfs data. For users who are not attaching data, it would be necessary to use the lfs$ prefix on both sides of the assign arrow (->) to tell R that we want the log of the variable HOURPAY

```
>      lfs$lnhourpay <- log(lfs$HOURPAY)
```

We can also create a completely new variable by assigning to it the value that we want. For instance, the following will create a new –free- variable called 'test' with a constant value of 1.

```
>      test <- 1
```



```
summary(HOURPAY)

#**************************************************************
# section 2
#************************************

lfs[6,80]
CONMON[6]
CONMON[ ,80]

summary(HOURPAY[HOURPAY>0])
summary(HOURPAY[SEX=='Female' & HOURPAY>0])

lnhourpay <- log(HOURPAY)

# storing the variable lnhourpay in the lfs data

lfs$lnhourpay <- log(HOURPAY)

# Creating the 'test' variable

test <- 1
```

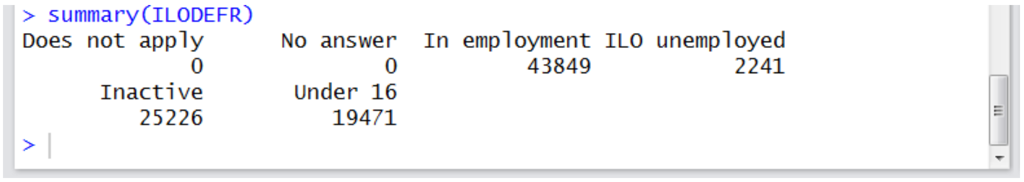
```
Data
lfs          90787 obs. of 769 variables
Values
lnhourpay    Large numeric (90787 elements, 709.3 Kb)
test         1
```

The new free variables are stored in the R environment under the tab 'Values'. You can also see that we now have 769 variables in the lfs data. The extra variable corresponds to 'lnhourpay' that we previously created.

5.3. Renaming variables and categories

In the case of categorical variables, we need to keep in mind that the possible values of the variables *are* what the summary() or the levels() commands returned – ie actual names: there are not numeric values to which value labels are added as with other programmes:

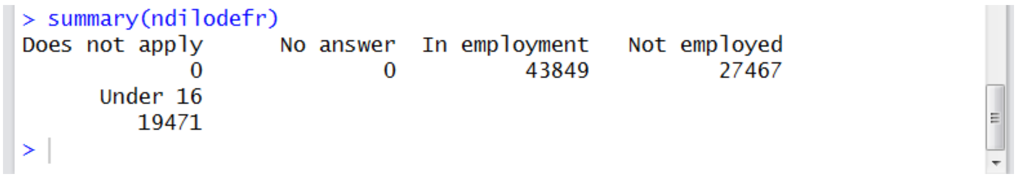
```
>       summary(ilodefr)
```



```
> summary(ILODEFR)
Does not apply      No answer  In employment ILO unemployed
             0              0          43849           2241
      Inactive       Under 16
         25226          19471
> |
```

The following commands will create a variable called ndilodefr where respondents in the 'Unemployed' and 'Inactive' categories of ilodefr are recoded as 'Not employed'.

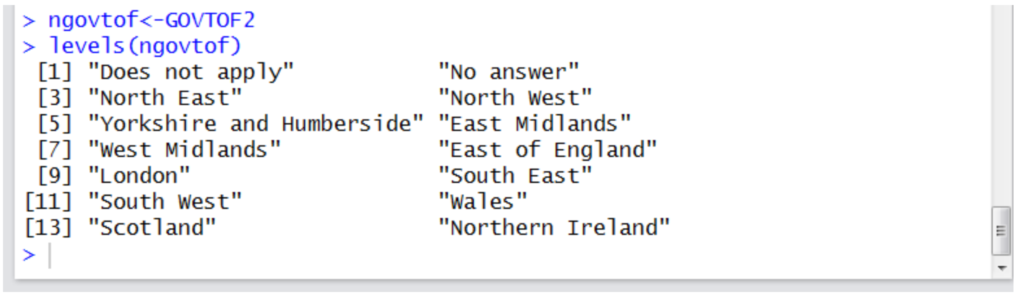
```
>       ndilodefr<-ilodefr

>       levels(ndilodefr)[4:5] <-'Not employed'

>       summary(ndilodefr)
```



```
> summary(ndilodefr)
Does not apply      No answer  In employment   Not employed
             0              0          43849          27467
      Under 16
         19471
> |
```

In order to alter the name of a variable or categories of categorical variables, one needs to directly rename the column of a data frame or the level of a factor.

```
>       ngovtof<-GOVTOF2

>       levels(ngovtof)
```



```
> ngovtof<-GOVTOF2
> levels(ngovtof)
 [1] "Does not apply"           "No answer"
 [3] "North East"               "North West"
 [5] "Yorkshire and Humberside" "East Midlands"
 [7] "West Midlands"            "East of England"
 [9] "London"                   "South East"
[11] "South West"               "Wales"
[13] "Scotland"                 "Northern Ireland"
>
```

```
>       levels(ngovtof)[5]<-'Yorkshire'
```

will abbreviate the name of the fifth category of the Government Office Region variable.

Note: The number [5] does not refer to the value attributed in the documentation of the variable, but to the number of the level in sequential order (in the example above, 6 refers to the sixth level in sequential order 'East Midlands').

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 otherwise specified (in our case, by adding the lfs$ prefix to variable creation command), the objects created are not included in the data frame from which they were computed. This should be kept in mind when combining such new variables and those within the original data frame. R might return error messages for example if missing values are coded differently in the two variables.

5.4. Missing values

Important: From now on, we will 'detach' the lfs file, so all the data manipulation will be stored in the actual data frame.

We first create a copy of the lfs data, so we will keep an intact copy of the original dataset. If you check the 'Environment' section of your R Studio screen, you will see two datasets loaded: lfs and lfs_copy

```
>       lfs_copy <- lfs
```

We will now detach the lfs file to avoid confusion. We will need to use the name of the dataset and the sign '$' every time that we refer to a variable.

This process seems to be unnecessary cumbersome, but fortunately R Studio has a very useful auto-completion function that allows us to avoid typing long fragments of code. This works for variables, datasets and functions. As usual, caution is needed to make sure that we are selecting the right variable.

```
>     detach(lfs)
```

By convention, missing observations in R are coded as 'NA'. Handling missing values in R is a bit more complex that in other packages as by default there are fewer safety nets than in other packages, for instance to tell users how many observations with missing values have been dropped in a variable. In addition, some commands can return error messages when dealing with variables whose missing values have been recoded to NA since they won't necessarily have the same number of 'rows'.

We can decide to recode missing values of hourly pay as NA, so that we don't have to select positive values each time we need to work with this variable. A simple way to do this is to force the value of NHOURPAY to NA if it is smaller than 0.

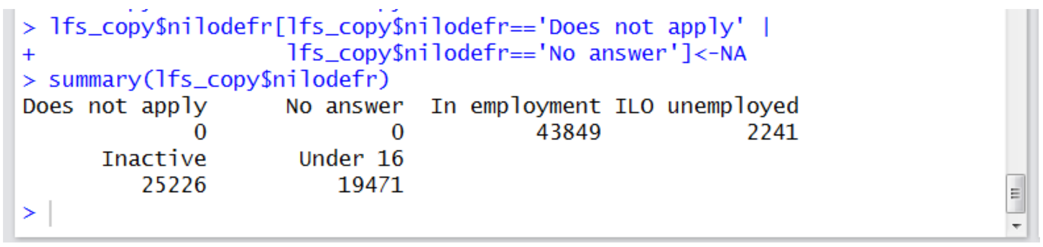
```
>     lfs_copy$nhourpay<-lfs_copy$HOURPAY

>     lfs_copy$nhourpay[lfs_copy$HOURPAY<0]<-NA
```

Users can recode values of either numeric objects or factors into missing values. A safe way to proceed is to create a new variable, which will contain the modified variable.

```
>     lfs_copy$nilodefr<-lfs_copy$ILODEFR
```

In order to keep things simple we will simply remove the levels of the factor that correspond to missing values.

```
>     lfs_copy$nilodefr[lfs_copy$nilodefr=='Does not
apply' | lfs_copy$nilodefr=='No answer']<-NA
```



```
> lfs_copy$nilodefr[lfs_copy$nilodefr=='Does not apply' |
+                   lfs_copy$nilodefr=='No answer']<-NA
> summary(lfs_copy$nilodefr)
Does not apply      No answer  In employment ILO unemployed
             0              0          43849           2241
      Inactive       Under 16
         25226          19471
> 
```

This method is not fully satisfactory since in some cases, such as contingency tables created with the table() or xtabs() commands, the values of ILODEFR that have been recoded as NA in NILODEFR will not appear anymore no matter what– one needs to keep a record of changes in the number of observations in the dataset.

Forcing values of variables to NA is not always necessary. In the case of categorical variables, there are commands which offer the option to mask categories that have no observation, which R doesn't do by default.

Once unwanted values have been recoded to NA, they can be taken care of by R's own missing values functions. In several commands, adding the na.rm parameter will prevent NAs from being shown (typing '?na.rm' will provide more information) . Depending on their needs, users can choose to use the option 'na.rm=T' (which tells R to remove missing values from an analysis).

An alternative is to use conditional statements when possible or to select a subset of the data:

```
>    lfs2 <- subset(lfs, ILODEFR!="Does not apply" &
         ILODEFR!="No answer" & ILODEFR!="Under 16"
         & GOVTOF2!="Does not apply" & GOVTOF2!="No
         answer" & HOURPAY>0 & SUMHRS>=0)
```

which will create a new data frame of 9,979 cases, where the unwanted values (in this case, the unused levels of ILODEFR, GOVTOF2, HOURPAY, SUMHRS) of the original LFS dataset will have been removed. This dataset is made of respondents who provided information about their region of usual residence, their economic activity and who were either effectively working or on temporary sickness / maternity leave or holiday during the reference week. Respondents who did not provide information about their earnings are also kept out.

Users will need to select the new data frame either by specifying it explicitly with the lfs2$ prefix before the name of a variable or alternatively by attaching it as the default dataset in memory.

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

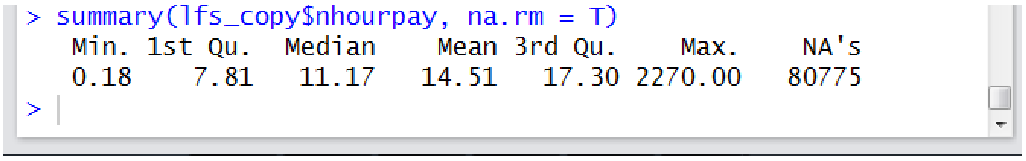
6. Descriptive statistics using R - continuous variables

In this section, we will review a few typical analyses that beginners may be interested in: producing uni- and bivariate statistics with either continuous or categorical variables.

Producing descriptive statistics in R is easy, some of the functions come ready to use without any additional package needed. We have already seen above that the summary() command provides essential information about a variable. For instance,

```
>       summary(lfs_copy$nhourpay, na.rm = T)
```

will give information about the mean, median and quartiles of the hourly earnings of respondents.



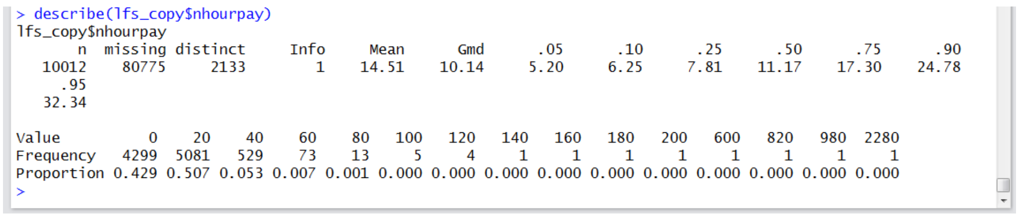
```
> summary(lfs_copy$nhourpay, na.rm = T)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
   0.18    7.81   11.17   14.51   17.30 2270.00   80775 
> 
```

The describe() command in the Hmisc package provides a more detailed set of summary statistics. We need to load Hmisc first

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

>       library(Hmisc)
```

Then

```
>       describe(lfs_copy$nhourpay)
```

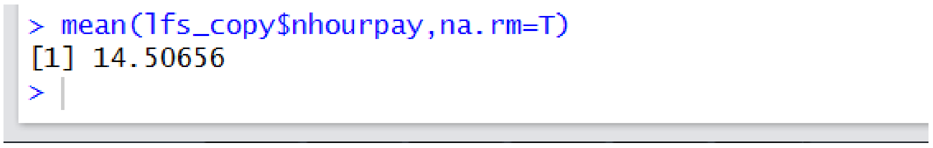


```
> describe(lfs_copy$nhourpay)
lfs_copy$nhourpay 
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75      .90 
   10012    80775     2133        1    14.51    10.14     5.20     6.25     7.81    11.17    17.30    24.78 
     .95 
   32.34 

Value          0    20    40    60    80   100   120   140   160   180   200   600   820   980  2280
Frequency   4299  5081   529    73    13     5     4     1     1     1     1     1     1     1     1
Proportion 0.429 0.507 0.053 0.007 0.001 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
> 
```

describe() also gives the number of observation (including missing and unique observations), deciles as well as the four largest and smallest values.

In addition to this, commands producing specific statistics are also available:

```
>       mean(lfs_copy$nhourpay,na.rm=T)
```



```
> mean(lfs_copy$nhourpay,na.rm=T)
[1] 14.50656
> 
```

The added na.rm = T (or na.rm = TRUE) option prevents missing values from being taken into account (in which case the output would have been NA). Other similar commands that compute individual statistics are available by default , such as sd(), max(), min().

Using these individual commands may come in handy, for instance when further processing of the result is needed:

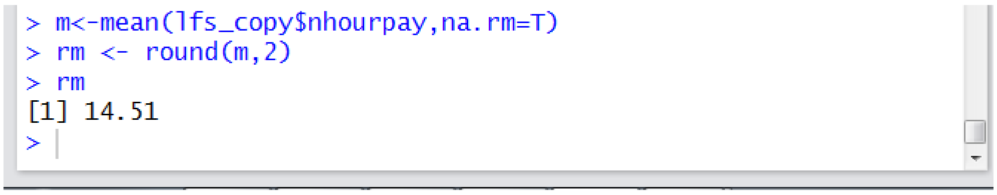
```
>       m<-mean(lfs_copy$nhourpay,na.rm=T)
```

Let's round the results to two decimal places:

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

We can see the final results by typing:

```
>       rm
```



```
> m<-mean(lfs_copy$nhourpay,na.rm=T)
> rm <- round(m,2)
> rm
[1] 14.51
> 
```

Note:

```
>       round(mean(lfs_copy$nhourpay,na.rm=T),2)
```

would have produced the same results and displayed it immediately on screen.

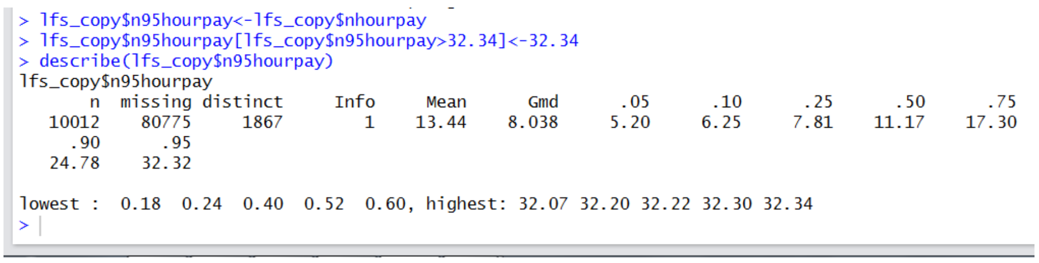
6.1 Distribution graphs

Another way to get a quick overview of the distribution of a variable is to produce a histogram, which is what the hist() command does. We will 'top-code' values of NHOURPAY over the 95th percentile in order for the histogram not to give too much importance to extreme values.

```
>       lfs_copy$n95hourpay<-lfs_copy$nhourpay

>       lfs_copy$n95hourpay[lfs_copy$n95hourpay>32.34]<-
32.34
```

Note: This could also have been achieved this way the quantile() function:

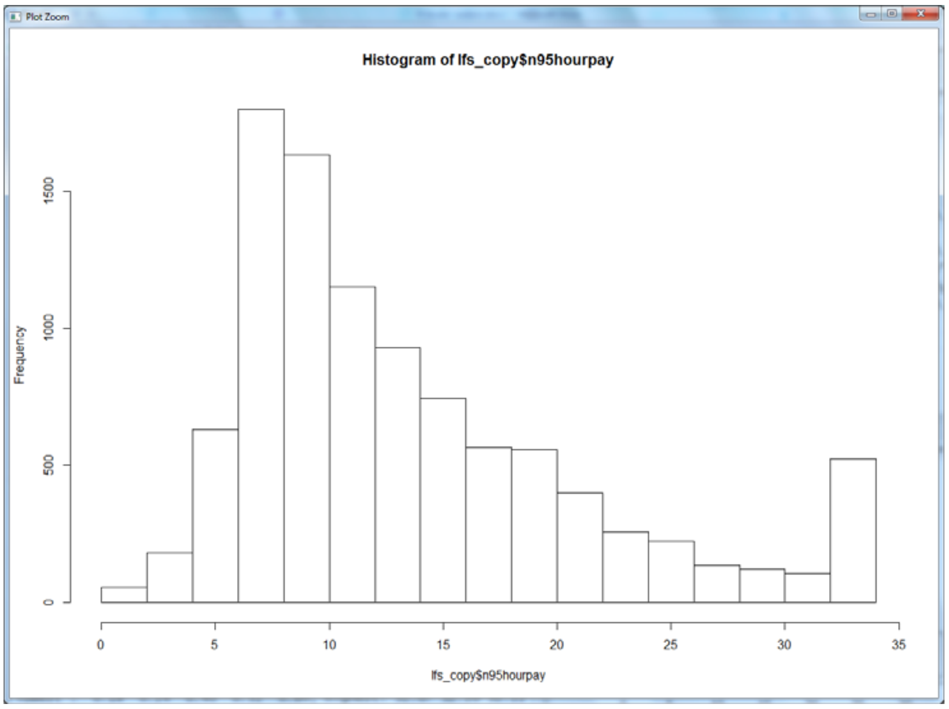
```
>       lfs_copy$n95hourpay[lfs_copy$n95hourpay>
quantile(lfs_copy$nhourpay,.95,            na.rm=T)]      <-
(quantile(lfs_copy$nhourpay,.95,  na.rm=T))

>       describe(lfs_copy$n95hourpay)
```



```
> lfs_copy$n95hourpay<-lfs_copy$nhourpay
> lfs_copy$n95hourpay[lfs_copy$n95hourpay>32.34]<-32.34
> describe(lfs_copy$n95hourpay)
lfs_copy$n95hourpay
       n  missing distinct     Info     Mean      Gmd      .05      .10      .25      .50      .75
   10012    80775     1867        1    13.44    8.038     5.20     6.25     7.81    11.17    17.30
     .90      .95
   24.78    32.32

lowest :  0.18  0.24  0.40  0.52  0.60, highest: 32.07 32.20 32.22 32.30 32.34
> 
```

```
>       hist(lfs_copy$n95hourpay)
```

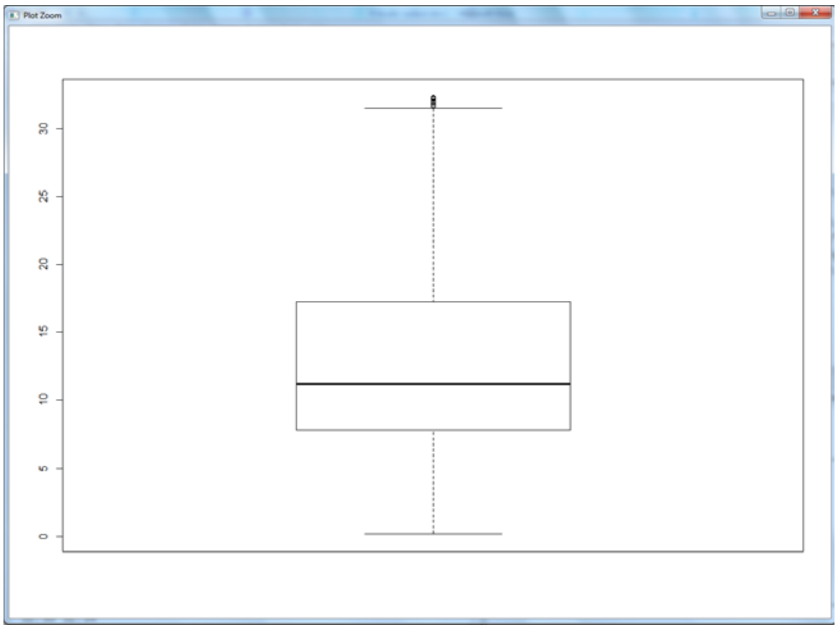
The histogram will be visible in the ‘Plot’ tab in the lower right side of the R Studio window. You can see it in another screen by clicking on ‘zoom’. There are other options available for the plots; you are free to explore them on your own.



Custom titles, notes and legends as well as colours, can be added to a R plot. See ?hist for more details.

Instead of a histogram, we can also produce a box and whisker plot of the same variable:

```
>       boxplot(lfs_copy$n95hourpay)
```



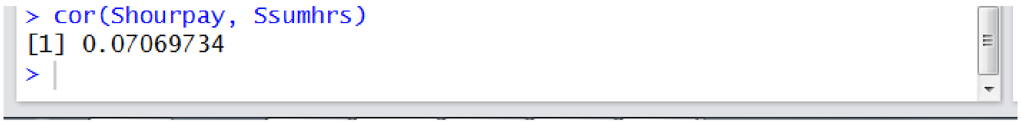
R also offers a wide range of bivariate statistics by default. In the same fashion as mean() above, corr(), cov() var()provide basic measures of association.

First, we need to create a new (free) variable for a subset of respondents with a valid answer in HOURPAY. We are selecting those respondents with HOURPAY values greater than 0 and SUMHRS values greater than or equal to 0. The new variable 'Shourpay' will be stored under the 'Values' tab (remember that R is case sensitive).

```
>    Shourpay<-lfs_copy$HOURPAY[lfs_copy$HOURPAY>0       &
     lfs_copy$SUMHRS>=0]
```

Now, we need to create an equivalent variable for a subset of respondents with a valid answer in SUMHRS. We are selecting those respondents with HOURPAY values greater than 0 and SUMHRS values greater than or equal to 0. The new variable is 'Ssumhrs'.

```
>    Ssumhrs<-lfs_copy$SUMHRS[lfs_copy$HOURPAY>0 &
     lfs_copy$SUMHRS>=0]


>    cor(Shourpay,Ssumhrs,use='complete.obs')
```



```
> cor(Shourpay, Ssumhrs)
[1] 0.07069734
>
```

We could have obtained the same results by using the 'lfs2' subset we created earlier:

```
>    cor(lfs2$HOURPAY, lfs2$SUMHRS, use='complete.obs')
```

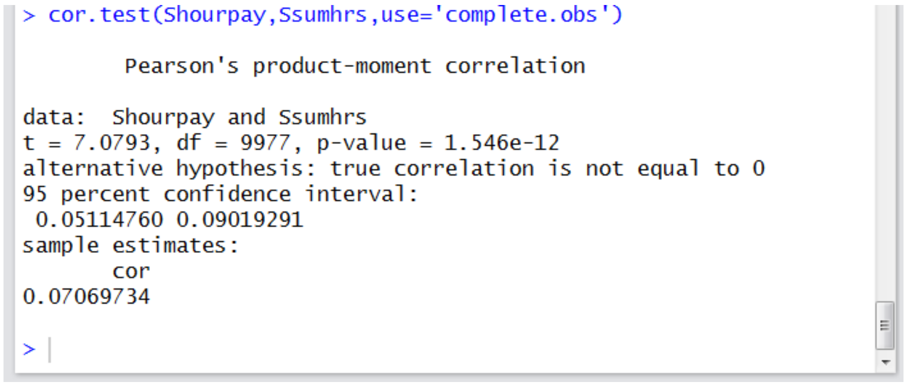
Note: cor() and cov() allows users to choose the method used for computing the correlation (between Kendall, Spearman, and Pearson) which enable running it on non-normally distributed variables (for example on ordinal variables).

Note: cor() and cov() (but not var()) return an error when na.rm is specified, instead of use=''. Users will need to refer to the documentation (by typing ?cor) for additional information.

6.2. Significance testing of correlation

We can perform a T test on the values returned by cor() with the cor.test(). cor.test()will also return a confidence interval:

```
>      cor.test(Shourpay,Ssumhrs,use='complete.obs')
```

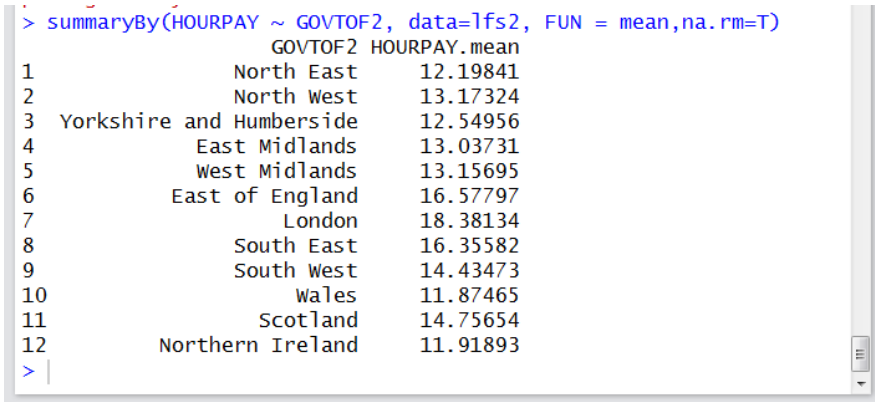


```
> cor.test(Shourpay,Ssumhrs,use='complete.obs')

        Pearson's product-moment correlation

data:  Shourpay and Ssumhrs
t = 7.0793, df = 9977, p-value = 1.546e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.05114760 0.09019291
sample estimates:
       cor
0.07069734

> 
```

6.3. Tables of summary statistics

We may want to go a bit further and produce a table of summary statistics, that is to inspect the distribution of a variable for (a) given subpopulation(s) or across the categories of another variable, for which we will need the command summaryBy() in the package doBy.

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

>      library(doBy)

>      summaryBy(hourpay   ~   govtof,   data=lfs2,   FUN   =
mean,na.rm=T)
```



```
> summaryBy(HOURPAY ~ GOVTOF2, data=lfs2, FUN = mean,na.rm=T)
                     GOVTOF2 HOURPAY.mean
1                 North East     12.19841
2                 North West     13.17324
3   Yorkshire and Humberside     12.54956
4              East Midlands     13.03731
5              West Midlands     13.15695
6            East of England     16.57797
7                     London     18.38134
8                 South East     16.35582
9                 South West     14.43473
10                     Wales     11.87465
11                  Scotland     14.75654
12          Northern Ireland     11.91893
> 
```

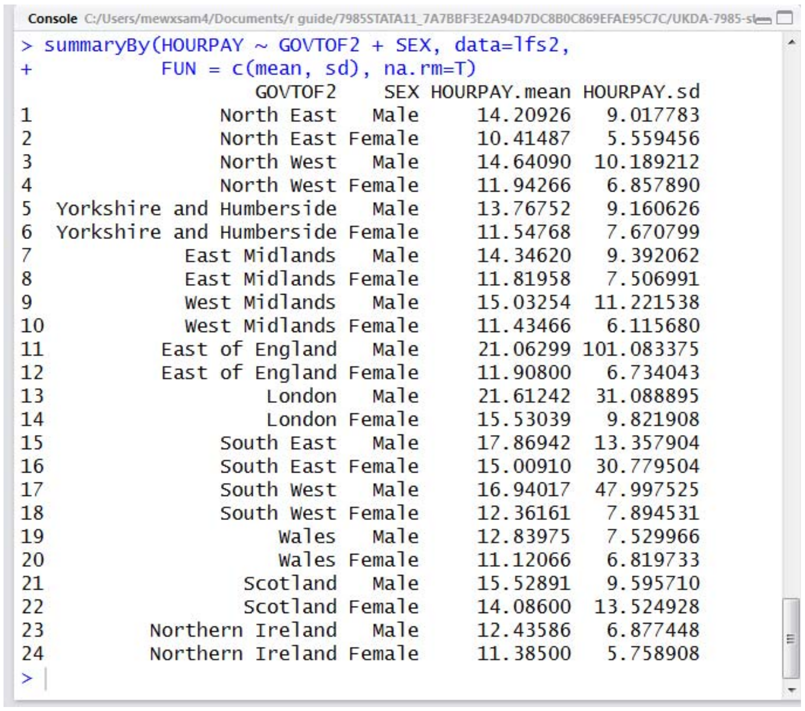
Notes:

The first parameter which specifies the continuous and categorical variable delimited by a tilde (~) is called a 'formula' in R jargon and is used by several other commands (such as contingency tables or regression analysis).

- In the case of summaryBy(), the second term of the formula is always the categorical variable, and there can be more than one specified, in which case the mean will be computed for each combined categories of the variables to the right of the tilde.
- The 'FUN' term can consist of several statistics, in which case they must be combined using the c() command.
- The source of the data needs to be mentioned explicitly whether or not a data frame has been previously attached as the default one.

A more complex version of the summaryBy() command looks like this:

```
> summaryBy(HOURPAY ~ GOVTOF2 + SEX, data=lfs2, FUN = c(mean, sd), na.rm=T)
```



```
Console C:/Users/mewxsam4/Documents/r guide/7985STATA11_7A7BBF3E2A94D7DC8B0C869EFAE95C7C/UKDA-7985-s
> summaryBy(HOURPAY ~ GOVTOF2 + SEX, data=lfs2,
+           FUN = c(mean, sd), na.rm=T)
                    GOVTOF2    SEX HOURPAY.mean HOURPAY.sd
1                North East   Male     14.20926   9.017783
2                North East Female     10.41487   5.559456
3                North West   Male     14.64090  10.189212
4                North West Female     11.94266   6.857890
5  Yorkshire and Humberside   Male     13.76752   9.160626
6  Yorkshire and Humberside Female     11.54768   7.670799
7             East Midlands   Male     14.34620   9.392062
8             East Midlands Female     11.81958   7.506991
9             West Midlands   Male     15.03254  11.221538
10            West Midlands Female     11.43466   6.115680
11          East of England   Male     21.06299 101.083375
12          East of England Female     11.90800   6.734043
13                   London   Male     21.61242  31.088895
14                   London Female     15.53039   9.821908
15               South East   Male     17.86942  13.357904
16               South East Female     15.00910  30.779504
17               South West   Male     16.94017  47.997525
18               South West Female     12.36161   7.894531
19                    Wales   Male     12.83975   7.529966
20                    Wales Female     11.12066   6.819733
21                 Scotland   Male     15.52891   9.595710
22                 Scotland Female     14.08600  13.524928
23         Northern Ireland   Male     12.43586   6.877448
24         Northern Ireland Female     11.38500   5.758908
> 
```

6.4. Bar charts and plots of summary statistics

As with any other R command, we can store the output of summaryBy() into an object. In this case, the default class is a data frame, with the same properties as the ones described above. The advantage of having results stored in a data frame is that it can be easily used with one of R graphical commands. These graphs can be saved as files which can in turn be imported in Word documents.

Let us begin with a simplified version of the summary table shown above and store it in a data frame called g:

```
> g<- summaryBy(HOURPAY ~ GOVTOF2, data=lfs2, FUN=c(mean,sd),na.rm=T)
```

There are several plots commands that we can use to plot these data: barplot() is the most common and works out of the box with simple plots (for instance if we only had one categorical variable in the data):

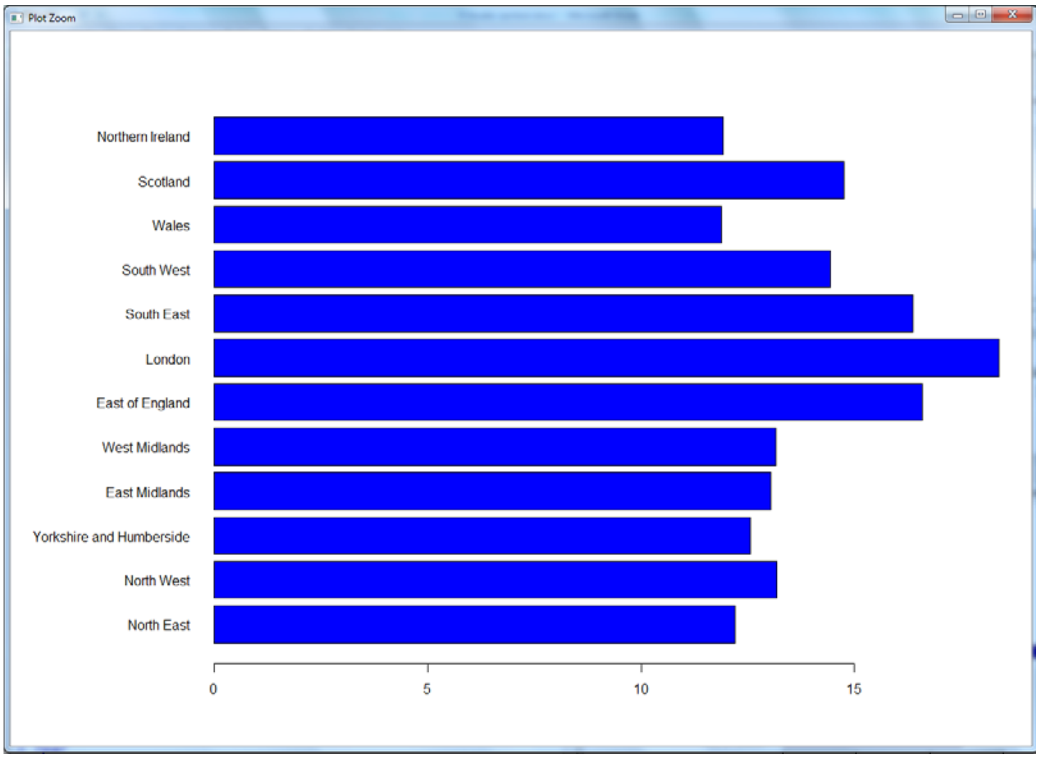
```
> barplot(g$HOURPAY.mean, names.arg=g$GOVTOF2, horiz=T)
```

The first parameter specified the variable to be plotted, the second one the labels or categories for each one of the means that are plotted. In both case, the data frame to which the values belong needs to be specified.

This initial plot is not fully satisfactory: the names of the regions are not displayed properly. Using a horizontal bar plot and adjusting the margins of the window would give a better result:

```
>     par(mar=c(5,12,4,2))

>     barplot(g$HOURPAY.mean,          names.arg=g$GOVTOF2,
horiz=T,

col='blue’, las=1)
```

The first command widens the space dedicated to the labels in the graph window, given the length of the regions’ names, whereas the las parameter allows labels to be displayed horizontally:



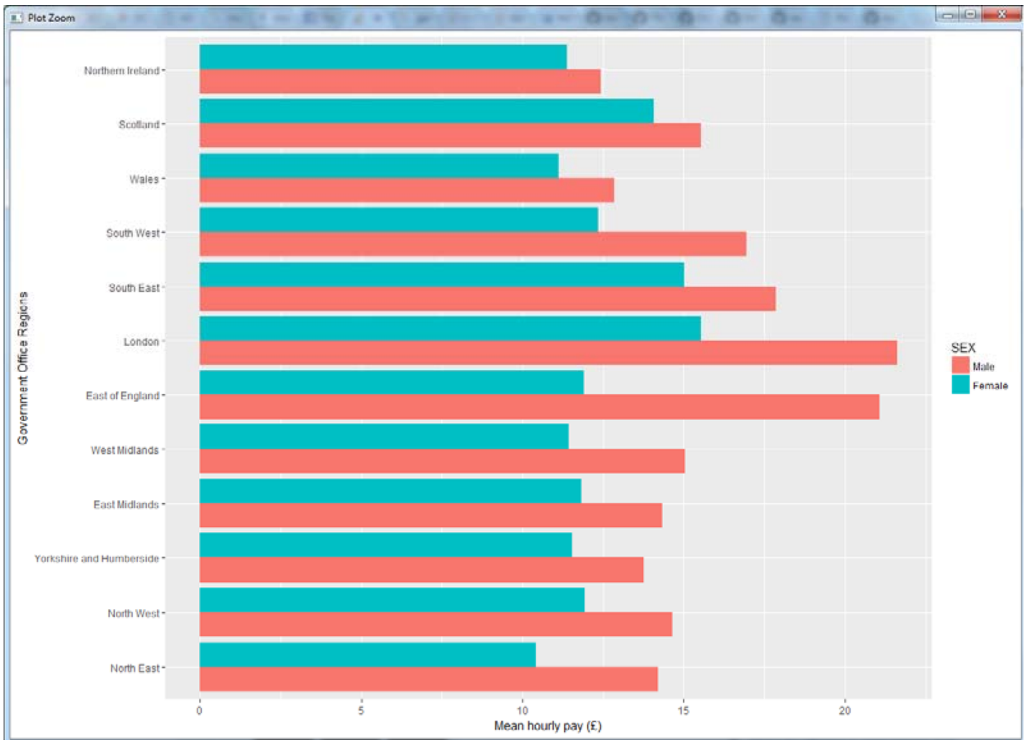
This graph still looks a bit rudimentary. The ggplot() command in the ggplot2 library will give a better output of publication quality. It is however more complex to use.

First, let us go back to our initial two way table of hourly earnings by gender and Government Office Region:

```
>     install.packages('ggplot2')

>     library(ggplot2)

>     g2<- summaryBy(HOURPAY ~ GOVTOF2 + SEX, data=lfs2,

FUN = c(mean), na.rm=T)
```

Now, we create the plot

```
>     ggplot(g2, aes(GOVTOF2, HOURPAY.mean, fill=SEX)) +

geom_bar(stat='identity', position = 'dodge') +

coord_flip() +
xlab('Government Office Regions') +
ylab('Mean hourly pay (£)')
```



Note: Unlike Stata, R does not assume that commands end with lines. It will continue reading the script file until it finds the characters that are supposed to complete it: in many cases, the closing bracket ')'.

Unfortunately, the syntax of ggplot2 does not fully respect the convention of most other R commands. Each parameter is specified by adding a '+', followed by the parameter name and its own options between brackets:

The first parameter is the name of the data frame from which the graph is to be drawn;

- aes() or aesthetic specifies the first two variables to be plotted as 'x' and 'y', whereas the third one is specified as 'fill' variable;
- geom_bar() specifies the appearance of the bars, for example whether stacked or not. Stacked is the default option; here we used 'dodge', which allows putting the categories of the 'fill' variable side by side;
- coord_flip() specifies the orientation of the graph (ie horizontal)
- xlab and ylab are the title of the x and y axis, respectively

ggplot offers many possibilities and it is recommended that users interested should consult the documentation on its website.

6.5 Saving and importing a graph in a Word document

Any R graph can be saved as an image file that can be subsequently imported in a Word or Latex document, for example. In this example, we will create a PNG image with the graph we have just created using the 'ggsave' option. By default, ggsave saves the last plot created, so we can simply write the following command, specifying the name of the plot (g2) and the format in which we want the graph. In this case, we will select 'png', but it can be other formats, such as: '.pdf', '.jpg', etc.:

```
>      ggsave('g2.png')

>      ggsave('g2.pdf')
```

We can also store the plot in R and then save the specific plot that we need. This is useful when we have several plots and we want some of them in a format that can be exported. With ggsave we can also specify the dimensions of our graph and the units of preference.

Here we save the plot in our R console under the name of 'g2plot'

```
>      g2plot<-   ggplot(g2,   aes(GOVTOF2,   HOURPAY.mean,
fill=SEX)) +

geom_bar(stat='identity', position = 'dodge') +

coord_flip() +

xlab('Government Office Regions') +

ylab('Mean hourly pay (£)')
```

Now we are saving (exporting) the plot into our working directory. The first argument, 'g2plot.png' corresponds to the name and format we are giving to our plot, the second argument is referring to the stored plot (in the R environment) that we want to save, and the others arguments indicate the dimensions and units of our png file.

```
>      ggsave('g2plot.png', g2plot,

        width = 15, height = 20, units = "cm")
```

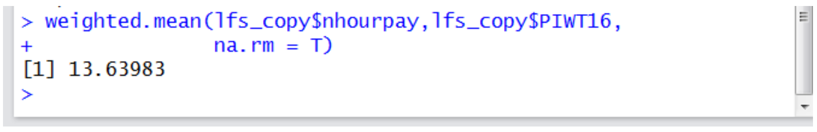
A more general approach to save any type of plot (without using ggsave) is the following, where the first line of command creates the file, the second one 'fills' it with a graph, and the third one, closes it.

```
>      png('g2plot2.png')

>      ggplot(g2, aes(GOVTOF2, HOURPAY.mean, fill=SEX)) +
geom_bar(stat='identity', position = 'dodge') +

coord_flip() +

xlab('Government Office Regions') +

ylab('Mean hourly pay (£)')

>      dev.off()
```

6.6 Weighted descriptive statistics

Most large UK surveys require sampling or grossing weights to be used in order to produce results that can be generalised to the population of interests. Some of the above R commands are designed to allow weights, such as weighted.mean(), a variant of mean().

```
>      weighted.mean(lfs_copy$nhourpay,lfs_copy$PIWT16,

               na.rm = T)
```



```
> weighted.mean(lfs_copy$nhourpay,lfs_copy$PIWT16,
+               na.rm = T)
[1] 13.63983
>
```

The table produced by summaryBy above (section 6.3) cannot be replicated using weights. Instead, we will need to use the dapply() function from the 'plyr' package.

Note: This requires installing the 'plyr' package, if not already installed.

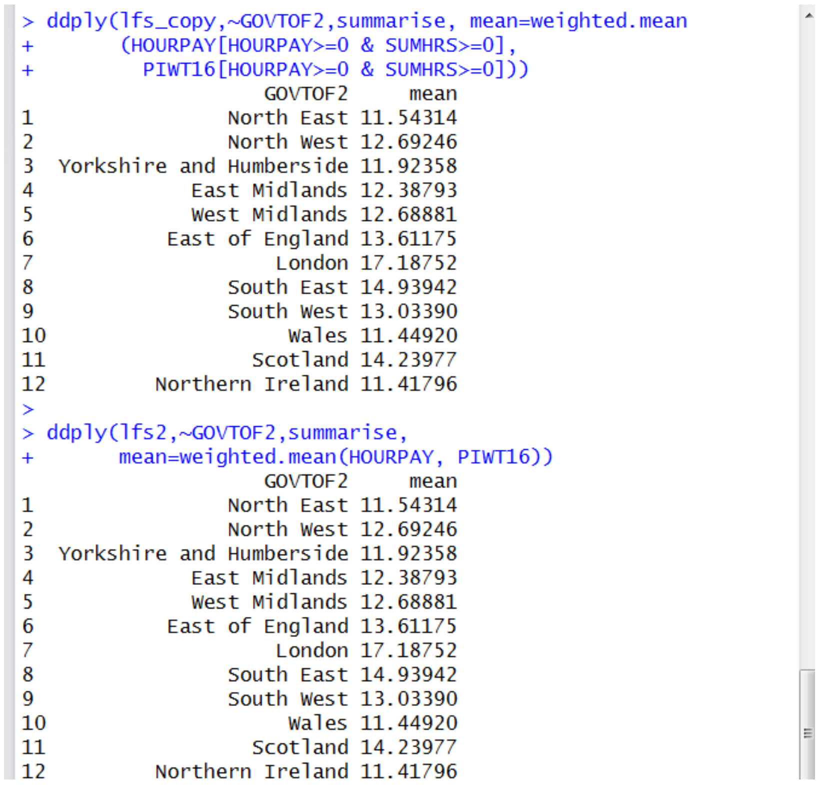
```
>      library(plyr)

>      ddply(lfs_copy,~GOVTOF2,summarise,
       mean=weighted.mean(HOURPAY[HOURPAY>=0 & SUMHRS>=0],

       PIWT16[HOURPAY>=0 & SUMHRS>=0]))
```

We could also use the subset created earlier:

```
>       ddply(lfs2,~GOVTOF2,summarise,

       mean=weighted.mean(HOURPAY, PIWT16))
```

For a given data frame(lfs2), and levels of one or several factors (ie categorical variables) the weighted mean of hourpay will be computed.

Note: The command requires the variable for which the mean need to be computed to be part of a data frame. The result is also a data frame, which lends itself well to drawing weighted graphs of variables means over multiple groups.


```
> ddply(lfs_copy,~GOVTOF2,summarise, mean=weighted.mean
+       (HOURPAY[HOURPAY>=0 & SUMHRS>=0],
+         PIWT16[HOURPAY>=0 & SUMHRS>=0]))
                    GOVTOF2     mean
1                North East 11.54314
2                North West 12.69246
3  Yorkshire and Humberside 11.92358
4             East Midlands 12.38793
5             West Midlands 12.68881
6           East of England 13.61175
7                    London 17.18752
8                South East 14.93942
9                South West 13.03390
10                    Wales 11.44920
11                 Scotland 14.23977
12         Northern Ireland 11.41796
>
> ddply(lfs2,~GOVTOF2,summarise,
+       mean=weighted.mean(HOURPAY, PIWT16))
                    GOVTOF2     mean
1                North East 11.54314
2                North West 12.69246
3  Yorkshire and Humberside 11.92358
4             East Midlands 12.38793
5             West Midlands 12.68881
6           East of England 13.61175
7                    London 17.18752
8                South East 14.93942
9                South West 13.03390
10                    Wales 11.44920
11                 Scotland 14.23977
12         Northern Ireland 11.41796
```

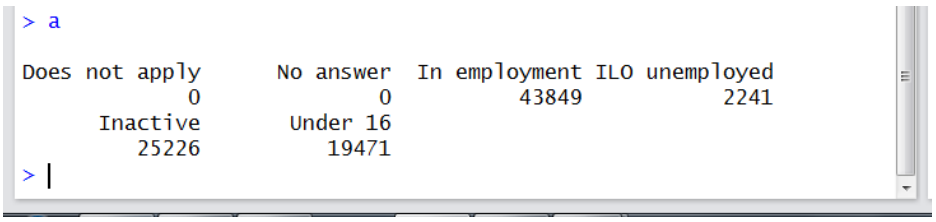
7. Categorical variables: contingency tables

As with continuous variables, R offers several tools that can be used to create contingency tables assess their statistical significance and graph the results.

7.1 One way frequency tables

The simplest R command that we can use is table() which returns the number of observations within each level of a factor:

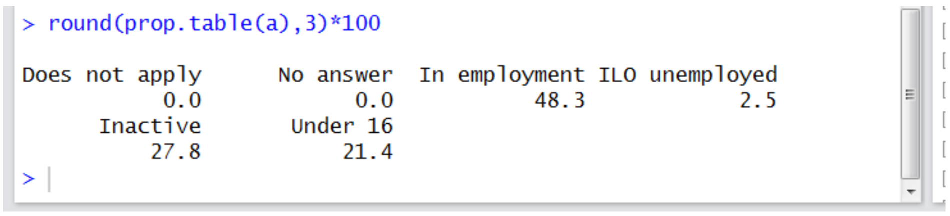
```
>       a<-table(lfs_copy$ILODEFR)

>       a
```



```
> a

Does not apply       No answer  In employment ILO unemployed 
             0               0          43849           2241 
      Inactive        Under 16 
         25226           19471 
> |
```

By itself, table() does not compute proportions nor percentages. These have to be specified manually using prop.table(). The following computes the percentages for the above table out of the raw proportions given by prop.table(). We also round the results to three decimal digits which actually means one digit once the proportions have been converted to percentages:

```
>       round(prop.table(a),3)*100
```



```
> round(prop.table(a),3)*100

Does not apply       No answer  In employment ILO unemployed 
           0.0             0.0           48.3            2.5 
      Inactive        Under 16 
          27.8            21.4 
> |
```

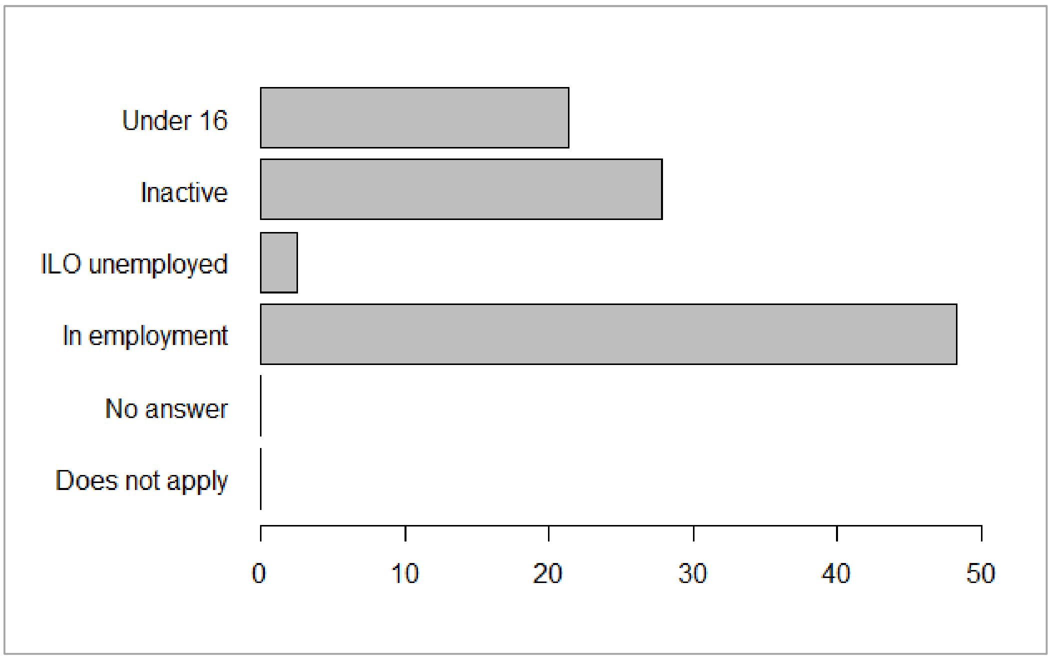
7.2 Creating bar plots of one-way frequency tables

The barplot() function described above is also suited to draw simple graphs of frequency tables. Going back to the previous command, we can type:

```
>       g3<-round(prop.table(a),3)*100

>       par(mar=c(4,8,2,2))
>       barplot(g3,horiz=T, xlim=c(0,50), las=1)
```

As above, we use par(), to adapt the size of the graph window to the length of category names of the ilodefr variable. The option ‘mar’ stands for margin size and represents a numeric vector of length 4, which sets the margin sizes in the following order: bottom, left, top, and right.

The result is shown next:



As with any plot() type of command, the colour of the bars, their orientation as well as titles can be easily specified – users can type ?barplot for more information.

7.3 Two way or more contingency tables

The table() command can also handle two-way contingency tables.

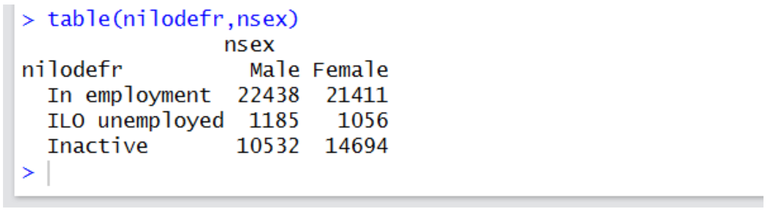
```
>       table(lfs_copy$ILODEFR, lfs_copy$SEX)
```

We can get a cleaner table using the recoded version of sex and ILO economic activity, following the logic described above. First, we recode non valid observations (ie, ‘Does not apply’, ‘No answer’,’ Under 16’ as system missing (NA):

```
>       nilodefr<-lfs_copy$ILODEFR

>       levels(nilodefr)[1:2]<-NA

>       levels(nilodefr)[4]<-NA

>       nsex<-lfs_copy$SEX
>     levels(nsex)[1:2]<-NA

>     levels(ngovtof)[1:2]<-NA
```

We can use table again to see obtain the cleaned table.

```
>     table(nilodefr,nsex)
```

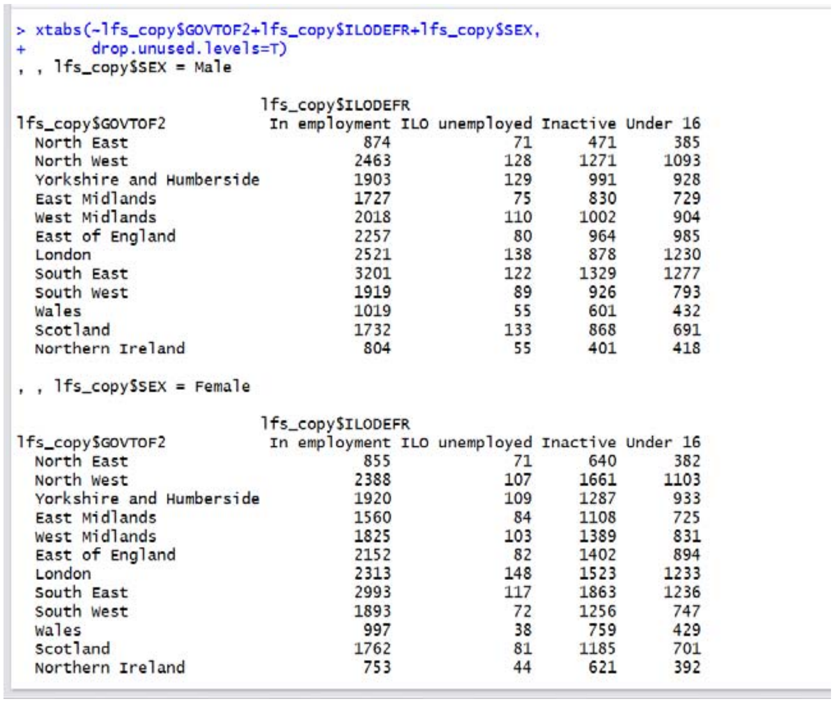


```
> table(nilodefr,nsex)
                nsex
nilodefr          Male Female
  In employment  22438  21411
  ILO unemployed  1185   1056
  Inactive       10532  14694
> 
```

If three or more variables are involved, then we will need to use another command. xtabs(), also available by default in R, allows for three way contingency tables. In order to examine regional differences in ILO economic activity, we can type:

xtabs() which follows a syntax that is similar to summaryBy()since the variables in the table are specified using a formula:

```
>     xtabs(~lfs_copy$GOVTOF2 + lfs_copy$ILODEFR +
lfs_copy$SEX, drop.unused.levels=T)
```



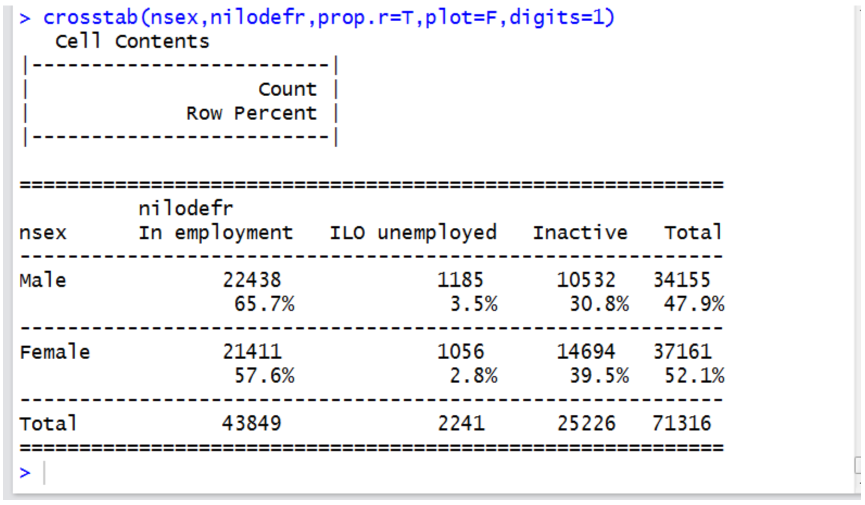
```
> xtabs(~lfs_copy$GOVTOF2+lfs_copy$ILODEFR+lfs_copy$SEX,
+       drop.unused.levels=T)
, , lfs_copy$SEX = Male

                          lfs_copy$ILODEFR
lfs_copy$GOVTOF2           In employment ILO unemployed Inactive Under 16
  North East                         874             71      471      385
  North West                        2463            128     1271     1093
  Yorkshire and Humberside          1903            129      991      928
  East Midlands                     1727             75      830      729
  West Midlands                     2018            110     1002      904
  East of England                   2257             80      964      985
  London                            2521            138      878     1230
  South East                        3201            122     1329     1277
  South West                        1919             89      926      793
  Wales                             1019             55      601      432
  Scotland                          1732            133      868      691
  Northern Ireland                   804             55      401      418

, , lfs_copy$SEX = Female

                          lfs_copy$ILODEFR
lfs_copy$GOVTOF2           In employment ILO unemployed Inactive Under 16
  North East                         855             71      640      382
  North West                        2388            107     1661     1103
  Yorkshire and Humberside          1920            109     1287      933
  East Midlands                     1560             84     1108      725
  West Midlands                     1825            103     1389      831
  East of England                   2152             82     1402      894
  London                            2313            148     1523     1233
  South East                        2993            117     1863     1236
  South West                        1893             72     1256      747
  Wales                              997             38      759      429
  Scotland                          1762             81     1185      701
  Northern Ireland                   753             44      621      392
```

Note: The output of table() and xtabs() can be stored into object of the 'table' class.

```
>     b<- xtabs(~lfs_copy$GOVTOF2 + lfs_copy$ILODEFR +
lfs_copy$SEX, drop.unused.levels=T)
```

The crosstab() command in the package 'descr' allows us to directly obtain a two-way contingency table with row and/or column percentages

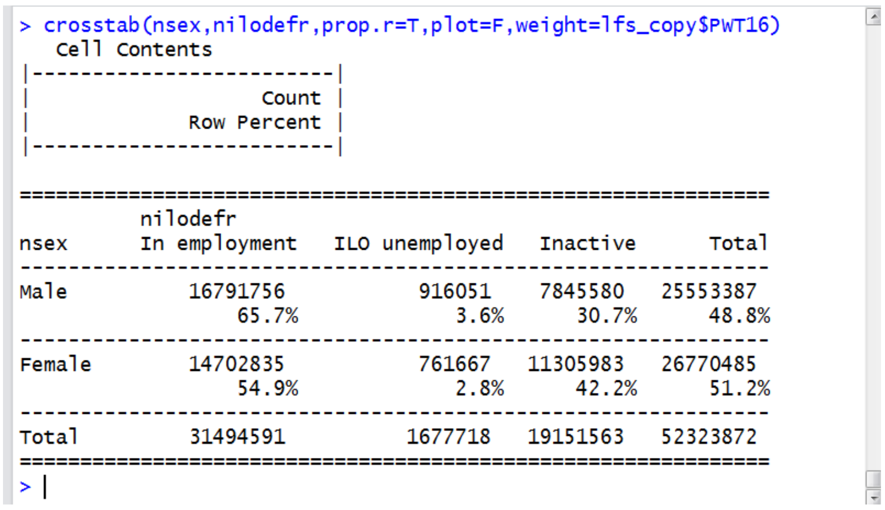
```
>      install.packages('descr')

>      library(descr)

>      crosstab(nsex, nilodefr, prop.r=T, plot=F, digits=1)
```



```
> crosstab(nsex,nilodefr,prop.r=T,plot=F,digits=1)
   Cell Contents
|-------------------------|
|                   Count |
|             Row Percent |
|-------------------------|

==========================================================
          nilodefr
nsex       In employment   ILO unemployed   Inactive   Total
----------------------------------------------------------
Male               22438             1185      10532   34155
                   65.7%             3.5%      30.8%   47.9%
----------------------------------------------------------
Female             21411             1056      14694   37161
                   57.6%             2.8%      39.5%   52.1%
----------------------------------------------------------
Total              43849             2241      25226   71316
==========================================================
> 
```

A useful feature of crosstab() is that it also allows observations to be weighted. We can thus produce the same command as above using the LFS weights PWT16:

```
>      crosstab(nsex,    nilodefr,    prop.r=T,    plot=F,
       weight=lfs_copy$PWT16)
```



```
> crosstab(nsex,nilodefr,prop.r=T,plot=F,weight=lfs_copy$PWT16)
   Cell Contents
|-------------------------|
|                   Count |
|             Row Percent |
|-------------------------|

==============================================================
          nilodefr
nsex       In employment   ILO unemployed   Inactive       Total
--------------------------------------------------------------
Male            16791756           916051    7845580    25553387
                   65.7%             3.6%      30.7%       48.8%
--------------------------------------------------------------
Female          14702835           761667   11305983    26770485
                   54.9%             2.8%      42.2%       51.2%
--------------------------------------------------------------
Total           31494591          1677718   19151563    52323872
==============================================================
> |
```

Note: Adding the option digit=1 (one decimal place), will make the table easier to read.

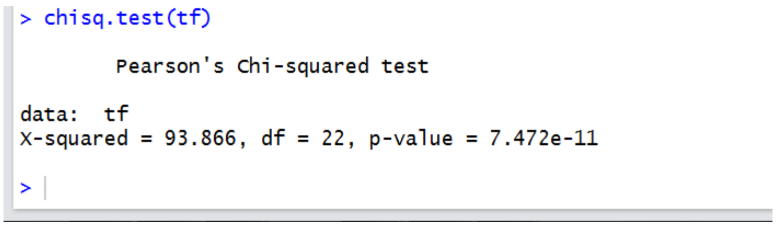
7.4. Test of association between categorical variables

We saw earlier that cor() allows us to compute Spearman and Kendall correlation coefficients, together with a significance test which provides a measure of association between ordinal variables. R also provides several ways to compute chi-square tests for contingency tables. In its simplest form, the chisq.test() command computes the Pearson's Chi-Square test for objects of the table class.

```
>     tf1<-table(nsex,nilodefr)
>     chisq.test(tf1)
```

Now, let's have a look at regional difference in economic activity for women only:

```
>     tf2<-table(ngovtof[nsex  =='Female'],  nilodefr[nsex
=='Female'])

>     chisq.test(tf2)
```

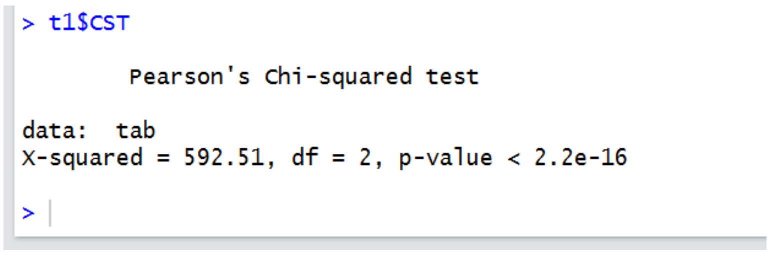


```
> chisq.test(tf)

        Pearson's Chi-squared test

data:  tf
X-squared = 93.866, df = 22, p-value = 7.472e-11

>
```

`crosstab()` provides its own version of the Pearson's Chi Square. It also provides McNemar's test and Fisher's Exact test.

The result of the test can be displayed immediately or stored in an object. The code below will reproduce the result for tf1.

```
>     t1<-crosstab(nsex,nilodefr,prop.r=T,plot=F,chisq=T)

>     t1$CST
```



```
> t1$CST

        Pearson's Chi-squared test

data:  tab
X-squared = 592.51, df = 2, p-value < 2.2e-16

>
```

An object of class 'CrossTable' contains a number elements (CST is one of them), some of which can be reused for further applications including graphs. Use summary to inspect the object ‘t1’ and use the dollar sing ($) after the object to access each one of them

```
>     class(t1)

>     summary(t1)

>     t1$tab
```

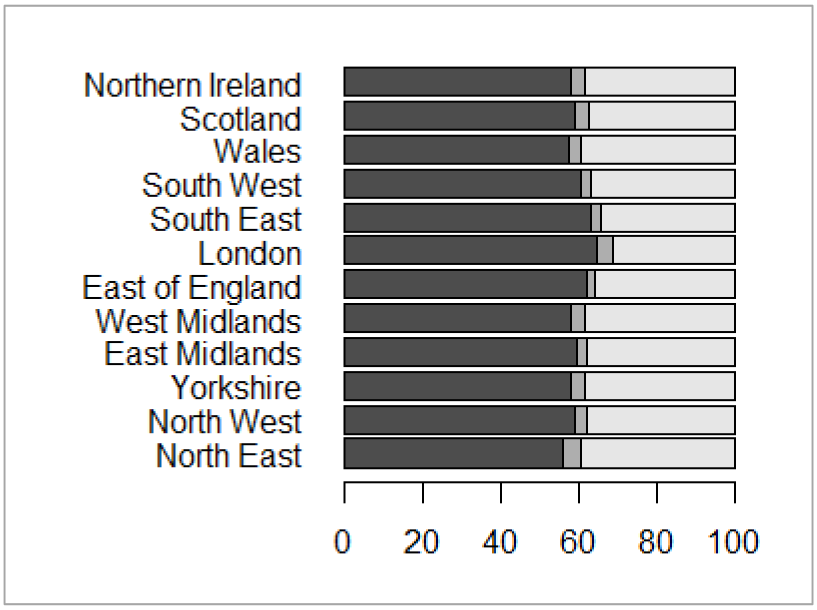
7.5. Univariate and bivariate graphs for categorical variables

Most of the graphical commands described above also allow us to compute graphs for categorical variables:

barplot() provides a simple way to plot the output from crosstab().

```
>     t2<-crosstab(nilodefr,       ngovtof,       prop.c=T,
      plot=F,chisq=T, lfs_copy$PWT16)
>     par(mar=c(5,10,4,2))

>     barplot((t2$prop.col*100), horiz=T,las=1)
```

We use the prop.col (ie column proportions) element of the crosstab output we called ‘t2’ to compute a graph of the percentages of respondents in each ILO economic activity category by Government Office Region:



As above in the case of continuous variables, barplot()provides a tool for the rapid visualisation of contingency tables. However, users who need more advanced graphic capabilities will use ggplot()and will refer to the package documentation for more information.

8. Plotting simple maps in R

This section presents a concise introduction on how to plot simple data (proportions, means, and other descriptive statistics) on a map. We will use the level of geography available in many UK Data Service datasets: Government Office Regions for England and Wales. We will use a contingency table of Government Office Region by economic activity and gender to build a map of weighted female employment rates by Government Office Regions for 2016.

Users need to check whether the following three packages have already been installed and if not, download and install them using the install.packages() and library() commands.

- rgdal
- tmap
- raster

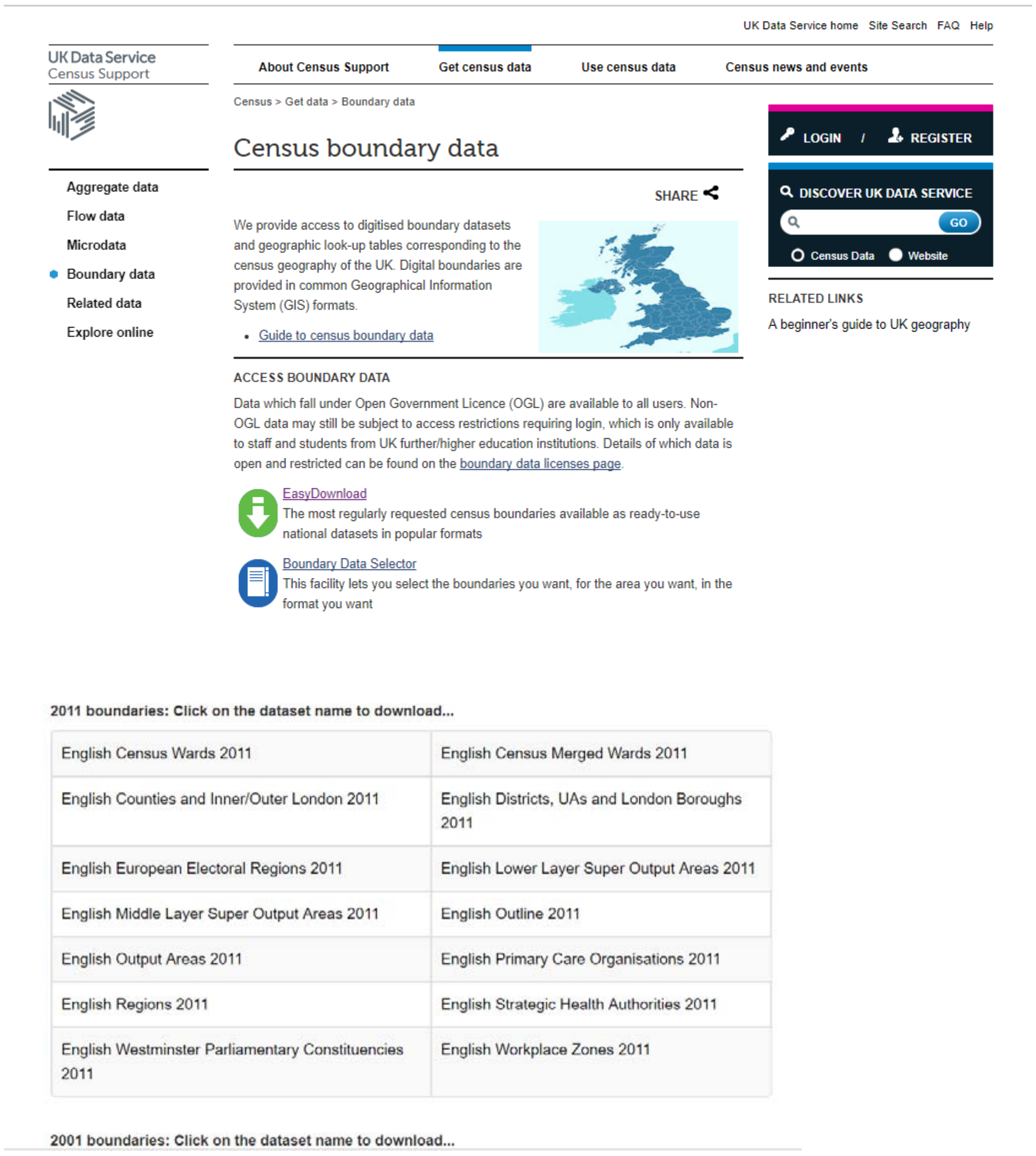
Let's begin by creating a new directory in order to keep our files tidy

```
>     dir<-"C:/Documents and Settings/INSERT YOUR USER NAME HERE/My Documents/R_ESDS/maps"

>     setwd(dir)
```

8.1. Acquiring and downloading the data

The files containing administrative boundary data that are necessary to create maps are available through the UK Data Service as part of Census Support. Users registered with a UK higher education institution are able to directly access the data via their institution’s login and password. Other users will need to register with the UK Data Service in order to obtain one.

You should go to the Census Support page for this data and click on the tab 'Get census data' and select 'Boundary Data'. After having logged in, you will need to scroll down the window and select the ‘Easy download’ link in order to access the main boundary data selection page.



Census boundary data

We provide access to digitised boundary datasets and geographic look-up tables corresponding to the census geography of the UK. Digital boundaries are provided in common Geographical Information System (GIS) formats.

- Guide to census boundary data

RELATED LINKS

A beginner's guide to UK geography

ACCESS BOUNDARY DATA

Data which fall under Open Government Licence (OGL) are available to all users. Non-OGL data may still be subject to access restrictions requiring login, which is only available to staff and students from UK further/higher education institutions. Details of which data is open and restricted can be found on the boundary data licenses page.

EasyDownload
The most regularly requested census boundaries available as ready-to-use national datasets in popular formats

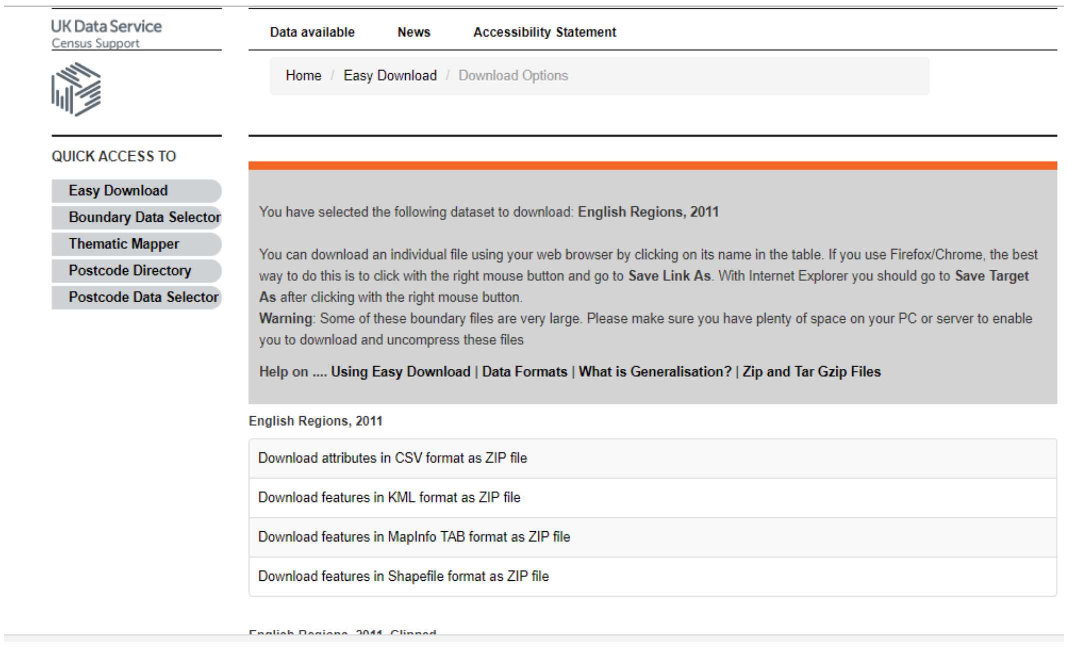
Boundary Data Selector
This facility lets you select the boundaries you want, for the area you want, in the format you want

2011 boundaries: Click on the dataset name to download...

English Census Wards 2011	English Census Merged Wards 2011
English Counties and Inner/Outer London 2011	English Districts, UAs and London Boroughs 2011
English European Electoral Regions 2011	English Lower Layer Super Output Areas 2011
English Middle Layer Super Output Areas 2011	English Outline 2011
English Output Areas 2011	English Primary Care Organisations 2011
English Regions 2011	English Strategic Health Authorities 2011
English Westminster Parliamentary Constituencies 2011	English Workplace Zones 2011

2001 boundaries: Click on the dataset name to download...

Click on ‘English Government Office Regions 2011’ and follow the instructions to download the files on your computer. You will need to select ‘Download features in Shapefile format as ZIP file’ as the file format.



You have selected the following dataset to download: **English Regions, 2011**

You can download an individual file using your web browser by clicking on its name in the table. If you use Firefox/Chrome, the best way to do this is to click with the right mouse button and go to **Save Link As**. With Internet Explorer you should go to **Save Target As** after clicking with the right mouse button.

Warning: Some of these boundary files are very large. Please make sure you have plenty of space on your PC or server to enable you to download and uncompress these files

Help on .... **Using Easy Download** | **Data Formats** | **What is Generalisation?** | **Zip and Tar Gzip Files**

English Regions, 2011

Download attributes in CSV format as ZIP file

Download features in KML format as ZIP file

Download features in MapInfo TAB format as ZIP file

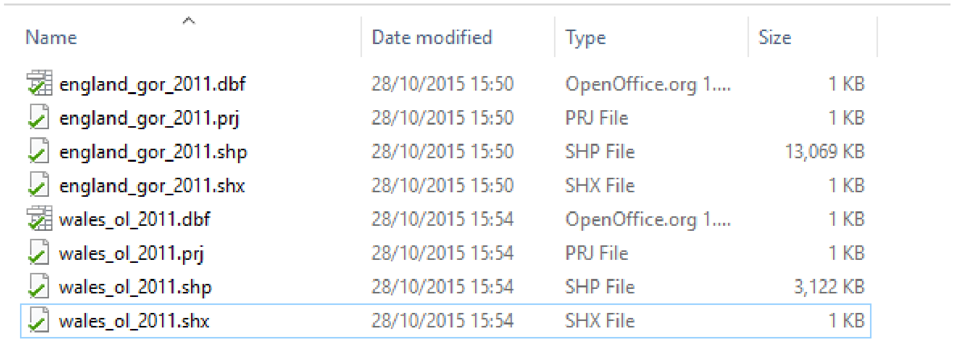
Download features in Shapefile format as ZIP file

The downloaded file should be called *England_gor_2011.zip*

Repeat this for Wales by clicking on the respective tab of the previous screen and select:

- 'Welsh Outline 2011' -> *Wales_ol_2011.zip*

Copy the zip files just downloaded into the newly created 'maps' directory and unzip them. There should now be 8 files in the directory:



Name	Date modified	Type	Size
england_gor_2011.dbf	28/10/2015 15:50	OpenOffice.org 1....	1 KB
england_gor_2011.prj	28/10/2015 15:50	PRJ File	1 KB
england_gor_2011.shp	28/10/2015 15:50	SHP File	13,069 KB
england_gor_2011.shx	28/10/2015 15:50	SHX File	1 KB
wales_ol_2011.dbf	28/10/2015 15:54	OpenOffice.org 1....	1 KB
wales_ol_2011.prj	28/10/2015 15:54	PRJ File	1 KB
wales_ol_2011.shp	28/10/2015 15:54	SHP File	3,122 KB
wales_ol_2011.shx	28/10/2015 15:54	SHX File	1 KB

8.2. Mapping in R – the basic principle

Creating a map in R can be decomposed into a few simple stages:

- collate together the boundary files ('Shapefiles') that we downloaded from the UK Data Service, in order to build a map of England and Wales

- add the data that we want to plot on the map
- decide on the intervals to be represented on the map, as well as the colour
- finally, we can plot the map itself

'Shapefiles' are usually made of a set of four actual files: .shp, .dbf, .shx, .cst. The first file type, '.shp', is the largest one and contains the geographical coordinates. They will be the ones we will be explicitly dealing with in this example. The other ones contain auxiliary information and are accessed by the mapping software in the background.

Given the limited scope of this example, detailed description of the Shapefile format will not be provided here. Users interested in learning, more about the specification of Shape files can consult this documentation on ESRI website.

An essential characteristic of shapefiles is that they are made of rows and columns in a similar fashion to conventional datasets. Columns are 'content' such as sets of geographical coordinates, area names and labels, or data that can be plotted on the map. Another one is that the maps that will be dealt with here are made of 'polygons', which represent the rows of a Shape file. A polygon is essentially a continuous area, the boundaries of which may be drawn without 'lifting the pen' (ESRI 1998). The mapping process described below essentially consists of filling these polygons with a colour or a symbol which represents the distribution of some variable we are interested in.

Since the shapefiles for England and Wales use the same reference system to draw the polygons (longitude and latitude coordinates), adding parts to an existing map is relatively straightforward. We can therefore append the maps for Wales with the map of England by copying it as an additional row into the Shapefile for England. The fact that the English dataset has internal boundaries (the Government Office Regions), whereas Wales does not, is not an issue here.

We will now move to the first stage of drawing our map

8.3 Producing the data needed for mapping

The command below produces a weighted three-ways contingency table of ILO economic activity by gender for each Government Office Region and country of the UK, which we will use to experiment with maps in this example.

Note: Specifying the weight *on the left hand side* of the formula is an alternative way of producing a weighted cross tabulation, which can deal with three way contingency tables, but does not provide a ready to use output in the same fashion as Crosstab() did as we saw before.

```
>    tmp<-xtabs(lfs_copy$PWT16~ngovtof+nilodefr+nsex)
```

For this example, only data for females are being selected

```
>    tf<-(tmp[,,2])
```

as.data.frame.matrix() below converts the output of the table (for which at the same time we compute the row percentages as we did in a previous section) into a data frame which can then be conveniently merged to the 'data' slot of the shapefile. In cases like this (output of table() type of commands), this is preferred to the more common as.data.frame(), which would not work properly.

```
>    tf<-
as.data.frame.matrix((100*round(prop.table(tf,1),3)))
```

We can remove the rows representing Scotland and Northern Ireland (rows 11 and 12 in the dataframe tf). This finalises the preparation stage of the data:

```
>    tf<-tf[-c(11:12),]
```

To add data to the Shapefile, we need to assign labels to each one of the GORs that will match those in the boundary file. We add a variable with the name of the region (so far these were the names of the rows of three data frames, not column/variables by themselves). This can also prove useful at a later stage to double check the results of the matching

```
>    tf$GOR<-row.names(tf)
```

We also need to change the name for 'Yorkshire' since the actual name of the region is 'Yorkshire and The Humber' and is also the name under which the region is identified in the shapefile.

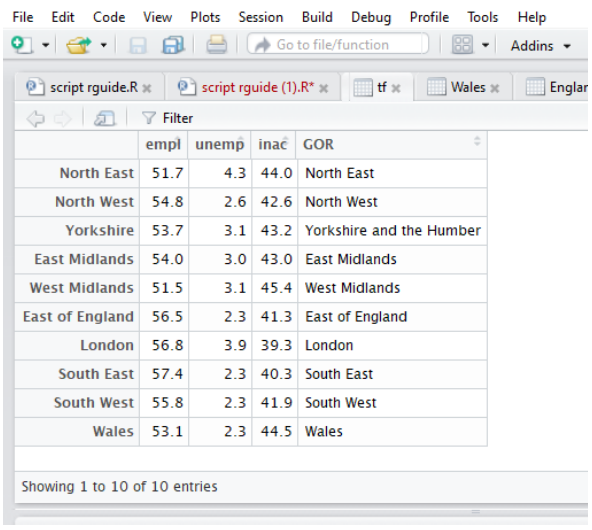
```
>    tf$GOR[tf$GOR=='Yorkshire']<-   “Yorkshire   and   The
Humber”
```

Let us abbreviate the variable names to make it easier when typing commands.

```
>    names(tf)[1:3]<-c('empl','unemp','inac')
```

Let us have a look at the final results

```
>    View(tf)
```



	empl	unemp	inac	GOR
North East	51.7	4.3	44.0	North East
North West	54.8	2.6	42.6	North West
Yorkshire	53.7	3.1	43.2	Yorkshire and the Humber
East Midlands	54.0	3.0	43.0	East Midlands
West Midlands	51.5	3.1	45.4	West Midlands
East of England	56.5	2.3	41.3	East of England
London	56.8	3.9	39.3	London
South East	57.4	2.3	40.3	South East
South West	55.8	2.3	41.9	South West
Wales	53.1	2.3	44.5	Wales

Showing 1 to 10 of 10 entries

8.4 Merging the boundary shapefiles and the data to be plotted

At this stage, we need to load the packages.

```
> library(maptools)

> library(tmap)

> library(rgdal)

> library(raster)
```

The following command from the package 'rgdal' will give you information about the shapefile, like the projection, the number of units, etc.

```
>     ogrInfo(dsn=dir,layer="england_gor_2011")

>     ogrInfo(dsn=dir,layer="wales_ol_2011")
```

Opening shapefiles is also a straightforward affair using the package 'rgdal'

```
>     England<- readOGR(dsn=dir, layer="england_gor_2011")

>     Wales<- readOGR(dsn=dir, layer="wales_ol_2011")
```

Note: This will return an error if the directory where the Shapefiles are stored was not properly specified.

The files are now stored in a special class of object called 'spatial polygons data frames'.

We can now append the spatial data frames together.

```
>     ew<- bind(England, Wales)
```

You can use the 'plot' function to see the new map created

```
>       plot(ew)
```

Now we need to merge the data from the contingency table with the 'ew' spatial data frame we have created. We first need to create the same variable 'GOR' of the contingency table in the shapefile; we need to convert the variable into a variable of class 'character', otherwise a warning would be issued.

```
>       ew$GOR<- ew$name

>       ew$GOR<-as.character(ew$GOR)
```

Then we use the 'left_join' function of the 'dplyr' package (install it first if you have not done so before)

```
>       library(dplyr)

>       ew@data<-left_join(ew@data, tf, by="GOR")
```

8.5. Plotting the map

We are ready to draw the map using the package 'tmap'. We will store our map it into an object called 'empFemale'.

```
>       empFemale<- tm_shape(ew)+

                    tm_polygons("empl",

                          title = "% employed females",

                          textNA="No data",

                          palette="Blues",        alpha=0.85,
                          breaks = c(-Inf, 52, 54, 56, Inf))+

                    tm_layout(legend.outside = F,

                          bg.color = "white")
```

In the above code, the first line specifies the Shapefile and the second line specifies the variable to be plotted. There are also more options available; here we only used some of the most useful, like:

'title' is used to give a title to the legend;

'textNA' specifies the label for missing data (in case you have missing values);

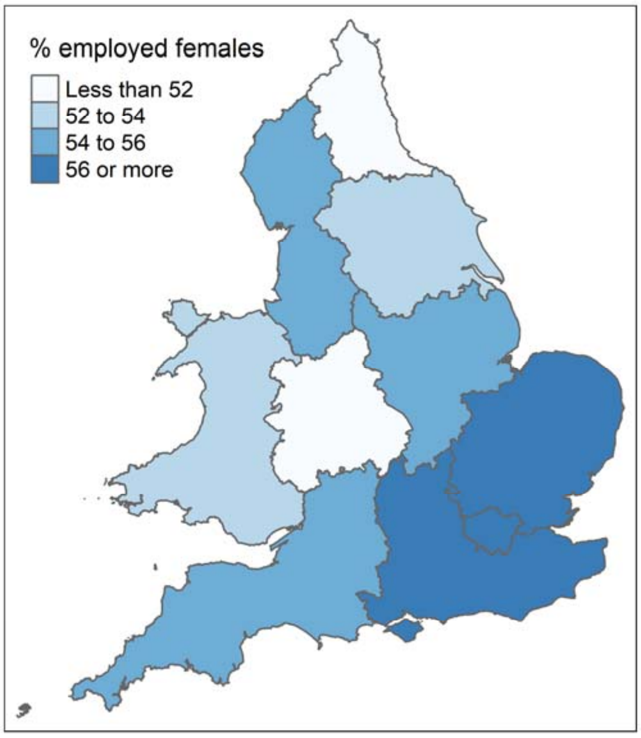
'palette' gives you the colour scheme;

'alpha' specifies the shade of the colour for the palette, 'breaks' is used to manually specify the intervals of the variable plotted.

We also used tm_layout to specify some options for the legend and background colour.

We can examine the resulting map by typing:

```
>       empFemale
```



Finally, as previously, we can save the results as a png file.

```
>       png("empFemale.png", width = 11, height = 10,

                  units = "cm", res = 600)

>       empFemale

>       dev.off()
```

This example can be expanded to build and map ad-hoc statistics with the data added to the Shapefile. For instance, by repeating the procedure described above with the contingency table of economic activity for men, one could map gender ratios of employment rates. Similarly, means or other descriptive statistics from continuous variables could be merged into the Shapefile instead.

9. Further commands and analyses

We have reached the limits of what can be illustrated within the space of this guide. Users interested in carrying out more advanced analysis should consult the links and references listed in the next section 'Additional resources'.

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

- **Regression analysis**: the glm() command installed by default with R can be used for fitting simple and multiple linear and non linear regressions including logistic regression and more generally models falling under the Generalized Linear Model framework. In addition, the package 'lme4' allows to fit linear multilevel (ie mixed effects) models, whereas 'nlme' allows to fit non linear multilevel 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..

- Users interested in **latent variable modelling** will be interested in the factanal() command. Other resources are provided in the 'poLCA' (Latent Class Analysis), 'ltm' (Latent Trait modelling), 'sem' (Structural equation modelling) packages

- 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. 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:

- www.ats.ucla.edu/stat/r/ - as with other statistical packages, the UCLA website provides a good starting point for the beginner
- www.unt.edu/rss/class/Jon/R_SC/ at the University of North Texas provides useful links to R resources
- www.r-bloggers.com/ contains many posts about R - in particular, www.r-bloggers.com/r-tutorial-series-r-beginners-guide-and-r-bloggers-updates/ contains useful introductory information
- stats.stackexchange.com/ is not specific to R but contains forum-type questions and answers raised by R users
- www.harding.edu/fmccown/r/ presents useful information about graphs in R.
- www.bristol.ac.uk/cmm/learning/course.html - 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