Data.frames: Checking out the data

20161214:10.00

On completion of this module you will:

- Be able to demonstrate an understanding of the importance of visualising your data sample to check whether or not it is representative of the population(s) that you wish to study
- Be able to demonstrate an understanding of the importance of finding out what characteristics the variables in the data sample are supposed to represent and how they have been derived
- Know how to view the data in any data.frame
- Understand and be able to demonstrate that you know how to use the summary function
- Understand and be able to demonstrate that you know how to use the basic table function
- Understand the implications of NAs when using the table function and demonstrate the ability to handle these
- Be able to set up age bands on any data.frame that has age and gender columns
- Be able to produce age / sex profiles on any data.frame, or any subset of a data.frame, that has age and gender columns
- Have developed further expertise in using row / column indexing
- Know how to use the nrow() function to count rows in a data.frame and how to use that function to keep a tally of rows that have been selected or deselected
- Be able to demonstrate an understanding of what is meant by 'data cleaning'
- Know at least two different ways of handling extreme values and NAs when cleaning numeric data
- Understand and be able to use logical operators to combine expressions to clean data
- Understand and know how to use the is.na() function
- Know how to set up simple histograms using the hist() function and where to go to find further details to refine them

Preamble

- It is suggested that you use <ALT> + <TAB> to alternate between this Workbook module and RStudio
- In this module where a series of commands open drop down menus or dialogue boxes we will provide the sequence to follow separated by the pipe "|" character (e.g. **Files |New File | R Script** to open up a new R script)
- This module assumes that you have already studied the modules **"Introducing RStudio"**, **"Creating vectors and working with them",** and **"Dataframes Indexing and Subsetting** or that you have a commensurate level of understanding of R and RStudio. You can review the learning content of these modules from the Workbook index pages. If you are in any doubt we would recommend that you study these modules first before proceeding any further
- For clarity we recommend that you clear your Environment and then reload the environment file **WBenv2.RData** that was saved at the end of the module **"Dataframes Indexing and Subsetting".** You should be able to find this in the **RStud-master** folder (see Initial setup below for details).

Initial setup

If you have worked through the preceding three modules you should already have the folder **RStud-master** in place and it should contain a file called **WBenv2.RData**. Provided that you have this RData file, continue as follows:

- Clear the Environment (Click here for help)
- Click on **Files** tab and navigate to the **RStud-master** folder
- Open this and set it as your Working Directory (hint – click on **More**)
- Click on the file **WBenv2.RData** and then on **"Yes"** to confirm
- The environment should now be set up as required

If you have not worked through the previous modules or if for any reason the RStud-master folder and / or the file WBenv2.Rdata are missing then select one of the following options:
Either drop back and work through the module "Dataframes, indexing and subsetting" and then return to this module

Or follow the instructions at the end of this link to obtain / recover the RStud_master folder and its contents. Having done this then continue as follows:

- Clear the Environment (Click here for help)
- Click on **Files** tab and if necessary navigate to the **RStud-master** folder
- Open this and ensure that it is set as your Working Directory (hint – click on **More**)
- Open the folder "**other**"
- In that sub folder find and Click on the file **WBenv2.RData** and then click **"Yes"** to confirm
- Return to your Working Directory Rstud_master
- The environment should now be set up as required

The importance of visualising the data

Researchers will almost always be carrying out their work on a data sample. The question therefore arises as to how representative this data sample is of the population that it is supposed to represent. This question should be addressed at an early stage and it should not be blindly assumed that the data is suitably representative and fit to support the research that is planned. It is important to look at the data and to check that it has realistic patterns of distribution for the various characteristics of interest. These patterns can if necessary be compared with those of data samples that have already undergone some form of validation. It is often far more effective to visualise the data than to depend solely on statistical tests.

For the purposes of this module we have a small data sample, with a very limited number of characteristics, held in the data.frame **WBdata**. It was randomly selected from a much larger database and the data in that larger database was itself obtained from a relatively small sample of English General Practices. If we were planning to use this data to test hypotheses about the population attending English General Practice it would be wise first to get a sense of how representative this sample is of the population attending English General Practice.

In this module we will demonstrate some of the ways that R can be used to visualise data samples and to provide a sense of the quality and representativeness of the data.

What variables does WBdata hold and what do they mean?

Start a new script and name this 'View and assess data sample'. Add and run the following line to this script:

```
str(WBdata)
```

The **str()** function confirms that the data.frame holds 10000 rows and has six variables named: Id, Age, GenderMF, Hypert, SBP, DBP
It tells us the data types for each of these variables and shows us some examples of the values. Note in particular that for both the SBP and DBP columns, which have data type 'int', there are values represented by 'NA'. This is the convention used by R to indicate that a value is 'Not Available' / unknown / missing.

What the **str()** function cannot tell us is what these columns and rows actually represent. That knowledge can only be obtained if we know from where the data was obtained, how it was obtained, and how it has been processed since that time. Thus separately from anything that R can tell us, and in addition to what R can tell us, we can provide the following information:

- Each row represents the data of one patient and no patient is represented by more than one row
- The Id column holds a unique pseudonym for each patient
- The Age column indicates the age of each patient on a given reference date
- The GenderMF column indicates whether patient gender is "F" for female or "M" for male
- The Hypert column indicates whether a patient has a recorded diagnosis of hypertension. "Yes" indicates diagnosis hypertension recorded. "No" indicates no record of diagnosis present. Note that we cannot tell how accurate or complete this diagnosis recording may have been
- The SBP column holds the latest record of systolic blood pressure expressed in mm of mercury
- The DBP column holds the latest record of diastolic blood pressure expressed in mm of mercury

Functions **head()**, **tail()**, and **View()**
These three functions can be helpful in viewing the data. Try them by adding and running the following script lines one by one:

```
head(WBdata)
```

```
tail(WBdata)
View(WBdata)
```

The head() function by default displays the first six rows of the data and the tail() function the last six rows of the data, in the console panel. Note that each of these can take a second numeric argument to display that number of rows. For example try:

```
head(WBdata,12)
```

The **View()** function displays the whole data.frame in the source panel and it is possible to scroll up and down, and for data.frames with many columns left and right, through the entire data.frame. Beware: this can be very slow for large data.frames with many columns:

```
View(WBdata)
```

Viewing the data by age and gender

A data sample will almost always include data for age and gender and clearly these two attributes can have a very significant effect on the distribution of other characteristics. For a sample to be representative of the original population it should therefore have a distribution of age and gender similar to that population. This is probably one of the most important checks. This can be done either by inspecting the data in the data.frame, summarising or by tabulating the data, or by viewing the plot of an age / gender profile.

Inspecting the data using **summary()** and **table()**

The functions **summary()** and **table()** can be helpful. Try each of the following lines in your script:

```
summary(WBdata$Age)
table(WBdata$Age)
summary(WBdata$GenderMF)
table(WBdata$GenderMF)
summary(WBdata$SBP)
table(WBdata$SBP)
```

Note the use of the $ naming convention here which was introduced in the previous module.

Note that the **summary()** function behaves differently depending on whether the column is of a numeric or categorical data type. When simply looking at numeric data it responds with Min, 1st quartile, Median, Mean, 3rd quartile, Max. When looking at categorical data it responds with an output which is very similarly to that of **table()**. It can only look at single columns and for that it is generally more useful than **table()**.

The **table()** function is of limited use when looking at single columns of numeric type because it tabulates the values found and the numbers with each value. However, it provides a useful output for single columns with categorical data type (e.g. see table(WBdata$GenderMF). It is important to note that with its default settings it ignores NAs (see below). The **table()** function can also create useful contingency tables by combining two columns. Try for example:

```
table(WBdata$Age,WBdata$GenderMF)
```

or better still:

```
table(WBdata$Age,WBdata$Hypert)
```

Note that the order of the arguments determines how the output is presented. The first argument dictates the output 'rows' and the second the 'columns'

Handling of NA

R denotes absent values / values that are not available with 'NA' and it is important to check how a given function /expression handles NAs. The **summary()** function will always show the count of NAs where these are present. However, beware; for the **table()** function the default is to ignore NAs. The easiest way to demonstrate the handling of NA by table() is by creating a small vector that contains some NAs: Add and run the following line to your script:

```
a <- rep(c(NA, 1/1:3), 10)
```

Note that **rep()** 'replicates' an expression – in this case ten times to produce a vector with 40 elements of which 10 will be NA. Use one of the 'help' mechanisms to find out more about **rep()**.

The **table()** function can take the "useNA=" argument. The values for useNA can be: "no", "ifany", "always". The default is "no" which results in no display of NAs. Try running the following lines in your script to explore the effects of setting the useNA argument to different values:

```
summary(a)                    #always shows NAs if present
```

```
table(a)                    #default shows no NAs

table(a,useNA="ifany")          #shows NA if present (as for summary)
table(a,useNA="always")          #always shows NA even if count is 0
table(WBdata$GenderMF,useNA = "always")
```

Note that in the last example the column GenderMF has no NAs but this is explicitly shown.

Suggested exercises:

- Re-run the last script line changing to useNA=”ifany”
- Use **summary()** and **table()** with useNA appropriately set to see which of the columns in WBdata has / has not got NAs

Viewing plots of age / gender profiles

Age / gender profiles, or age / sex profiles are typically presented using 5 or sometimes 10 year age bands and then plotting numbers of males and females in each band alongside each other. Therefore as a first step we will need to add a column to the WBdata data.frame which allocates each row to an ageband based on the value in the Age column. Needless to say, it is possible to do this in R but the process involves some 5 to 6 steps and the use of some 5 or 6 new functions. Therefore, to save time and avoid unnecessary complication we have produced a User defined function called **AddAgeBandsCol()** to encapsulate this process. Run the following line from your script:

```
WBdata2 <- AddAgeBandsCol(WBdata,AgeCol = "Age",5,20)
```

This assigns the resulting data to a new data.frame called WBdata2

Note that this function can be used with any data.frame that has a column holding age in years (note that this column can have any name). The function has four mandatory arguments:

- **df** for name of source data.frame
- **Agecol** for name of the column holding age in the source data.frame (needs to be inside quote marks)
- **AgeInterval** for ageband size in years – typically 5 or 10

- **IntervalCount** number of age bands (last band picks up all remaining higher ages)

There is a fifth optional argument – AgeBandName. If this is not set then the name of the ageband column will be set to AgeBands. Note that, as with any function, with the help of the <TAB> key you can add each of these argument names in full (e.g. Agecol="Age") and then the order of the arguments is not critical. Alternatively, so long as the arguments are in the correct order you simply need to enter their values – as in the example given

Run the following lines from your script:

```
str(WBdata2)
table(WBdata2$AgeBands,WBdata2$GenderMF,useNA="ifany")
```

The last line displays the number of males and females in each age band

The final step is to plot this data and once again we have produced a user defined function **PlotAsp()** to simplify this. Run the following line from your script:

```
PlotAsp(WBdata2,"AgeBands","GenderMF","Sample population")
```

A barplot Age sex profile should appear in your Plots panel (bottom left). This plot can be compared with those derived from other data samples to judge whether or not the age / sex distribution is representative

The **PlotAsp()** function will work with any data.frame so long as it has columns holding agebands and gender – and these columns can have any name. The function has four mandatory arguments:

- **popdf** for name of source data.frame
- **Agecol** for name of the column holding agebands in the source data.frame (needs to be inside quote marks)
- **Gendercol** for name of the column holding gender in the source data.frame (needs to be inside quote marks)
- **mtitle** for Main title above resulting barplot

Note that there is a possible extension to the capability of this barplot. We can use row / column indexing [,] to limit the sample to those patients who have hypertension. To do this we want to limit the sample to those rows where the Hypert column has the value "Yes". We can build this in steps. Go back to your script and copy the line:

```
PlotAsp(WBdata2,"AgeBands","GenderMF","Sample population")
```

We will add to this in steps. Try running the line after each step:
PlotAsp(WBdata2[,],"AgeBands","GenderMF","Sample population")

Then:
PlotAsp(WBdata2[WBdata$Hypert=="Yes",],"AgeBands","GenderMF","Sample population")

And finally:
PlotAsp(WBdata2[WBdata$Hypert=="Yes",],"AgeBands","GenderMF","Sample population with hypertension")

The expression WBdata$Hypert=="Yes" uses the $ naming convention to select the Hypert column and the ==”Yes” convention to look for all rows with the value “Yes”. Therefore the final PlotAsp line only ‘sees’ rows where it is true that the value in column Hypert is “Yes”.

Clearly the age sex profile for people with hypertension in this sample is very different from that for the whole sample. Is that what we would expect?

Subsetting again

Note that if we wanted, we could use the same approach to create a data.frame, called for example Hypert, that held only those patients with recorded diagnosis Hypertension. Try adding and running this line in your script:

Hypert <- WBdata2[WBdata$Hypert=="Yes",]

Exercise: Try using the PlotAsp() function to produce age sex profile of this sample. How does this compare with the age sex profile for hypertension created directly from the original sample?

Use of **nrow()**

When we extract a subset, or for any reason exclude data from a sample, it is useful to determine what proportion of the original sample has been affected. One simple way of doing this is by the use of the **nrow()** function. This will work on a vector or a data.frame. It counts the number of rows. We can do:

```
nrow(WBdata2)
```

```
nrow(Hypert)
```

What proportion of the original sample has hypertension? We can find this by doing:

```
s1 <-nrow(WBdata2)
s2 <-nrow(Hypert)
s2/s1 *100        #percentage of original sample with hypertension
```

Does that rate look realistic?

Age sex profile further refined

In the last section we looked at the age sex profile for patients with hypertension and also obtained an overall prevalence. However, it is clear that people with hypertension tend to be much older than the rest of the population. How does the prevalence vary with age and gender? There is a third user defined function, **PlotPrevAsp()**, that can provide a barplot of prevalence per 1000 of the original sample looking at each individual age / sex band. In your script try:

```
PlotPrevAsp(WBdata2,Hypert,"AgeBands","GenderMF","Hypertension")
```

The arguments for this function are exactly the same as for the **PrevAsp()** function except that there is an extra argument, second in order – called testdf for the name of the data.frame holding the subset (in this case Hypert)

Cleaning numeric data

Our data sample contains the two columns SBP and DBP which represent respectively latest systolic and diastolic blood pressures. This kind of data often contains extreme values that do not make any biological sense. Even more commonly it may contain instances of NA. It is worth taking a careful look at such data and then to plan what to do about extreme values and NAs. The process of dealing with extreme low values, extreme high values and NAs is often described as 'data cleaning'.

As we have already seen the summary() function can be useful in providing an overview here. For example we can do:

```
summary(WBdata2$SBP)
```

```
summary(Hypert$SBP)
```

This shows that there are proportionately far more NAs in the original sample than in the Hypert subset. Arguably there are no implausible extreme values. Removing the records from the original sample would lead to a loss of more than 40% of the rows. Clearly this could have a very significant impact on studies of any other characteristics that we might want to study in a larger and more varied sample. In contrast removal of the NAs from the Hypert subset would only lead to a loss of 1% of the rows.

Data cleaning can be done by using row / column indexing. In this case we would need to combine expressions for lowest permissible value, highest permissible value, and the handling of NAs. For the sake of providing an exercise when looking at the original sample we might decide to reject records with SBP less than 75 and greater than 235 – and to preserve NAs. We would need the following expressions: Try running these one by one to observe the results:

```
cleandf <- WBdata2$SBP > 74    #SBP values greater than 74
cleandf <- WBdata2$SBP < 236  #SBP values less than 236
```

The **is.na()** function

We also need to be explicit about NAs because otherwise R will not be able to evaluate NAs against these upper and lower limits resulting in rows containing all values set to NA. To retain NAs we can use:

```
cleandf <- is.na(WBdata$SBP)    #Retains all instances of NA
```

To reject NAs we can use the ! operator for “NOT”:

```
cleandf <- !is.na(WBPdata$SBP) #Retains all instances that are NOT NA
```

We can then concatenate all of these:

```
WBdata2$SBP > 74 & WBdata2$SBP < 236 | is.na(WBdata$SBP)
```

Note that in the above expression “&” is logical “AND” and the “|” character stands for logical “OR”. We want values to be higher than 74 AND lower than 236 OR to be “NA”. Putting this all together try adding the next line to your script and running it:

```
cleandf <- WBdata2[WBdata2$SBP > 74 & WBdata2$SBP < 236 |
is.na(WBdata$SBP),]
```

if we wanted to remove NAs we would change the above to:

```
cleandf <- WBdata2[WBdata2$SBP > 74 & WBdata2$SBP < 236 &
!is.na(WBdata$SBP),]
```

The above can seem very daunting to start with but it is worth experimenting with these expressions to develop a clear understanding of what is going on. However, to avoid hampering progress, we have provided a further user defined function **DataClean()** to simplify the above. This function has four mandatory arguments and one optional.

popdf	name of data.frame holding sample data
Datacol	name of column holding numeric data to be cleaned
lower	Values must be greater than this lower value
upper	Values must be less than this upper value

The fifth and optional argument is naremove. This defaults to TRUE. Where naremove=TRUE all records with NA will be removed. If naremove is set to FALSE then all records with NA will be retained

To carry out the data cleaning example above we would simply need to do:

```
cleandf <- DataClean(WBdata2,"SBP",74,236,naremove = FALSE)
```

This is clearly a great deal simpler than putting together the three expressions using "&" and "|" along with **is.na()** and "!". However, we would encourage you to spend some time doing things the 'long way' in order to have a better understanding of what is going on.

Exercise: Repeat all of the above section but using DBP instead of SBP – and setting appropriate upper and lower bounds.

Viewing numeric data

The histogram provides a useful and simple way of viewing numeric data. In your script try the following lines in your script:

```
hist(WBdata2$SBP)
hist(Hypert$SBP)
```

The function **hist()** has many arguments that can be used to modify and / or enhance appearance. We will provide some examples of the simpler ones here

but recommend that you use help to find out more about this useful function. Try adding the following lines to your script and then running them:

```
hist(WBdata2$SBP,xlim=c(0,250),breaks=seq(0,250, by=5),
col="red",main="Systolic BP in original data sample", xlab="BP in mm Hg")


hist(Hypert$SBP,xlim=c(0,250),breaks=seq(0,250, by=5),
col="red",main="Systolic BP in hypertensive subset", xlab="BP in mm Hg")#
```

Finally, take a look at:

```
summary(WBdata2$SBP)
summary(Hypert$SBP)
```

Compare the means, medians, 1st and 3rd quartiles. Are they what you might expect?

Exercise: Repeat the above for DBP

The module "Data.frames Checking out the data" finishes here.

Before closing down RStudio we would recommend that:

- You save your R script file for future reference
- Save your resulting environment to a new RData file called WBenv3.RData for future use (click here for help)

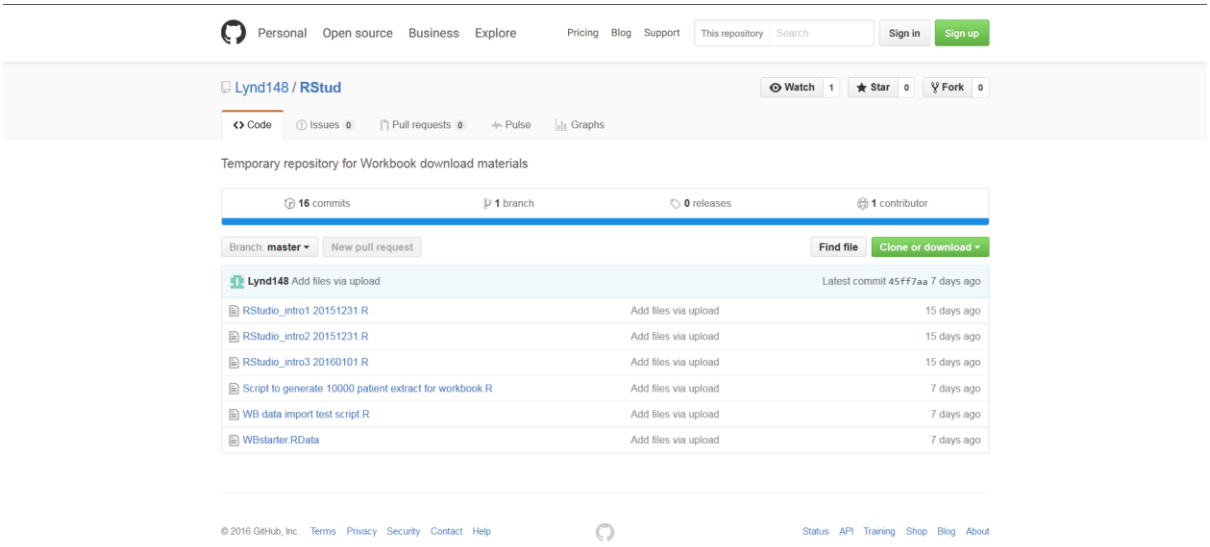
Instructions for setting up the environment

Download the scripts and Rdata resources from the Course web repository by selecting the appropriate link below
[Version for Desktop / notebook based RStudio]
[Version for Server based RStudio]

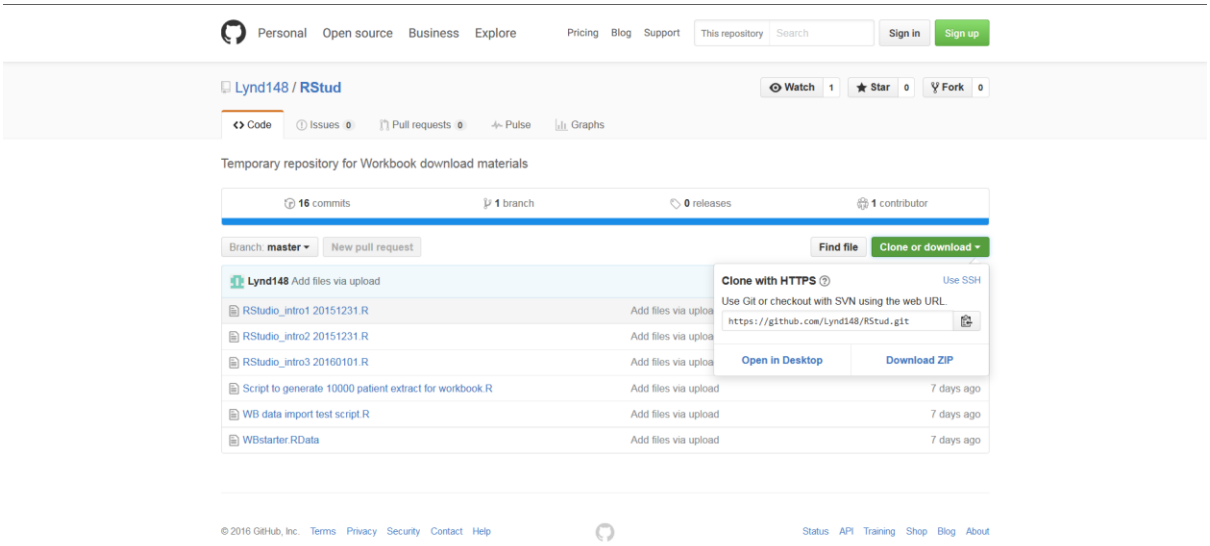
File download for Desktop / notebook based RStudio

1. Open you browser and either copy and paste the following URL to the address bar or type it in: **https://github.com/Lynd148/RStud.git**

2. This should open up a github repository with the title “Temporary repository for Workbook download materials” which will look something like this and contain a number of files. Note the button marked “Clone or Download” below the blue line



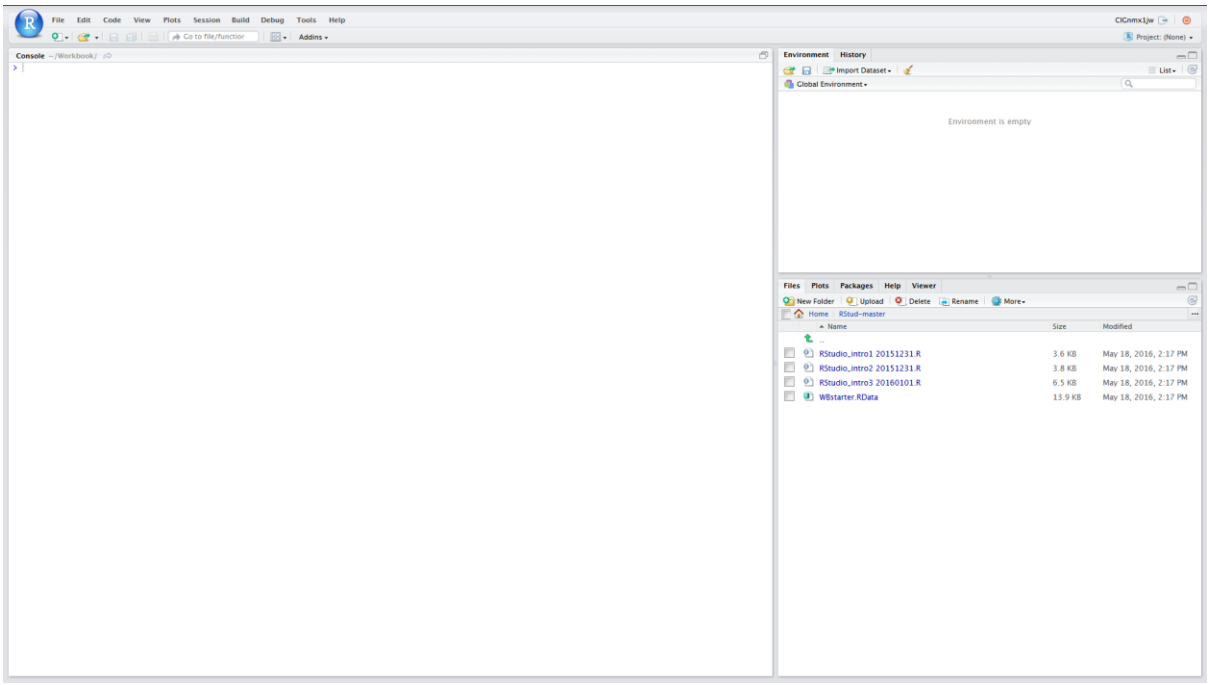
3. Click on the “Clone or Download” button. A dialogue box will open with two options: “Open in Desktop” and “Download ZIP



4. Click on the Download ZIP button. Depending on your browser and settings this will either open a dialogue box asking whether you want to open or save a file called RStud-master.zip or else will result in this file being immediately downloaded to your download folder. In the former case select ‘open’ and if necessary change the “Open with” option to Windows Explorer then click “OK”. The folder RStud-master should appear after being downloaded

   In the latter case (where the file has automatically downloaded) navigate to your download folder. Right click on the file and select ‘open with’ and the option Windows Explorer. The folder RStud-master should now appear
5. Copy the folder RStud-master to your C-drive or other easily found destination
6. Now return to RStudio. Make sure that the Files tab of the bottom right Files Plots Packages Help Viewer panel is active and click on the small box at the right edge of the screen containing three dots …

7. This should open a “Browse for folder” box. Browse to the folder RStud-master on your C-drive or wherever else you may have placed it and click OK.
8. The folder will now open under the Files tab which should look similar to the screen shot below



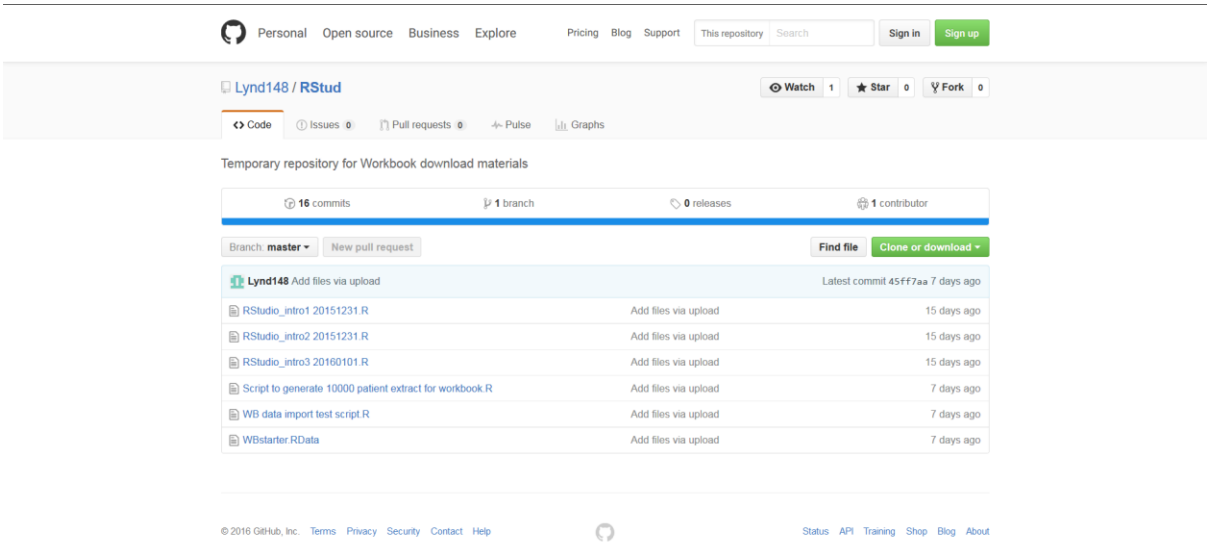
9. We now just need to set up the Working Directory. To do this click on the “More” tab (next to blue cogwheel icon just below the Files Plots Packages Help Viewer bar) and then Click on “Set as Working Directory”
10. All of the scripts and other files that we need in order to proceed should now be downloaded and also you have set the working directory
11. Click on the file WBstarter.RData to load it into the environment
12. Now save the environment to **WBenv1** click on the Environment tab and then the blue floppy disk item, enter **WBenv1** into the dialogue box and click Save

Click here to return to start of this module

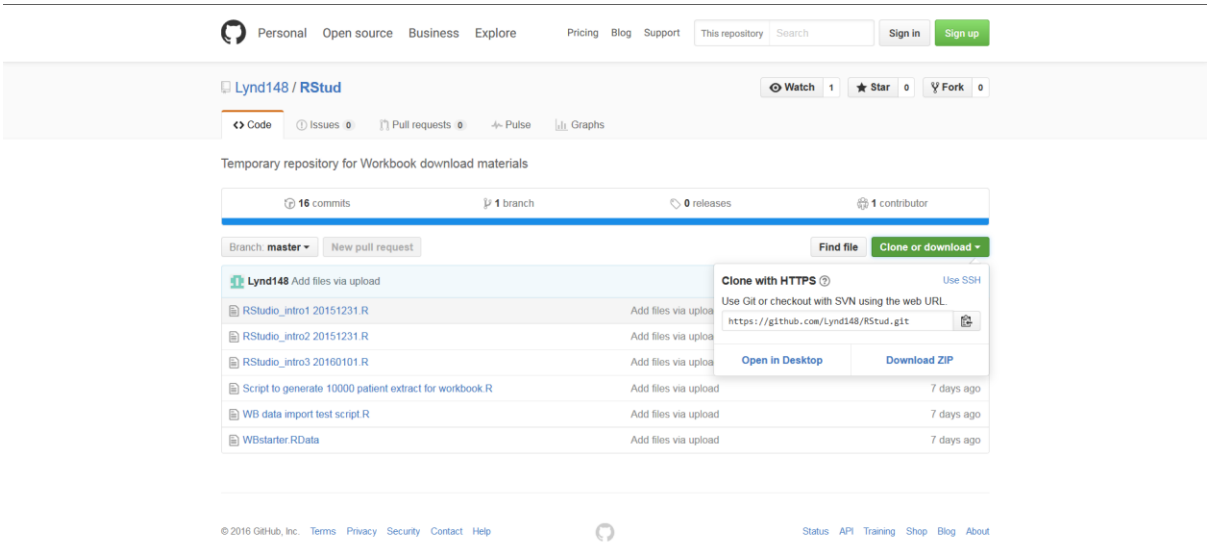
File download for server based RStudio

1. Open you browser and either copy and paste the following URL to the address bar or type it in: https://github.com/Lynd148/RStud.git

2. This should open up a github repository with the title “Temporary repository for Workbook download materials” which will look something like this and contain a number of files. Note the button marked “Clone or Download” below the blue line



3. Click on the “Clone or Download” button. A dialogue box will open with two options: “Open in Desktop” and “Download ZIP”



4. Click on the Download ZIP option. Depending on your browser and settings this will either open a dialogue box asking whether you want to open or save a file called RStud-master.zip or else will result in this file being immediately downloaded to your downloads folder. In the former case select 'save' and click "OK" because in this scenario we want a ZIP file to be placed in your download folder
5. Having ascertained the location of your downloads folder return to RStudio. Make sure that the Files tab of the bottom right Files Plots Packages Help Viewer panel is active.
6. Next you need to set your home directory to be the Working Directory
    a. Click on Home
    b. Click on More (next to blue cogwheel icon) and then Click on "Set as Working Directory"
7. Now look for the Upload button just below the Files Plots Packages Help Viewer bar. Click on this
8. This should open a "Upload Files" box. Within that box browse to the destination folder for your downloads and find the file RStud-master.zip and select / open this and then click on OK
9. This should result in the folder RStud-master being added to your home directory
10. Open this

11. Next you need to reset your home directory to this RStud-master directory. As before, click on the "More" tab (next to blue cogwheel icon) and then Click on "Set as Working Directory"
12. All of the scripts and other files that we need in order to proceed should now be downloaded and also you have set the working directory.
13. Click on the file WBstarter.RData to load it into the environment
14. Now save the environment to **WBenv1** click on the Environment tab and then the blue floppy disk item, enter **WBenv1** into the dialogue box and click Save

Click here to return to start of this module

Lookup

Set Working directory
File | More | Set as Working Directory
Click **<ALT> + Left Arrow** <- to return

To clear the environment:
Click on the Environment tab (upper left panel)
Click on the broom icon
In the Confirm Remove Objects dialogue box click 'Yes'
Click **<ALT> + Left Arrow** <- to return

To set up colnames vector:
colnames <- c("Id","Age","GenderMF","Hypert","SBP","DBP")
Click **<ALT> + Left Arrow** <- to return

Write out a data.frame to CSV file
If you use 'help' on the function **write.csv** you will see that it takes a number of arguments. With the exception of the first two arguments the remainder are set to default values. Therefore as a minimum, this function needs:

- As its first argument the name of the data.frame to be written out
- As its second argument the name of the file to which the contents should be written out
- Data.frames have row names as well as Column names and the default is for both of these to be written out. The assumption is that the first row will provide the column names – which in this case we want. We do not want the row names which at the time that a data.frame is created typically default to a rendition of the row numbers. Hence the argument row.names=FALSE

Click **ALT> + Left Arrow** <- to return

Help with row / column indexing

To limit the population to males only, use:

```
WBdata$GenderMF=="M"
```

To limit the population to age over 60, use:

```
WBdata$Age>60
```

To limit the population to those with hypertension, who are male, and over 60 years age, use "&" operator to combine expressions:

```
Hypertdf <- WBdata[WBdata$Hypert=="Yes" &
WBdata$GenderMF=="M" &
WBdata$Age>60,]   #line 3
```

For more detail about logical operators see **Introducing RStudio** module

Click **ALT> + Left Arrow** <- to return

Subsetting using the R function **subset()**

A web search on 'R subset' will reveal this function and instructions about its use. It can make the creation of subsets much easier than by using [,] row column indexing.

This function has been deliberately omitted from this module for two reasons. Firstly, one the objectives of this module is to explain row / column indexing. Secondly, expert R users sound caution in its use and indicate that its arguments (for example 'select') can be confusing and that in certain settings it produces unexpected results. See for example: http://stackoverflow.com/questions/9860090/in-r-why-is-better-than-subset

We would therefore advocate using row / column indexing rather than **subset()**. If you do use the **subset()** function – then do so with caution

Click **ALT> + Left Arrow** <- to return

Use of **rm()**
This function can handle a list of objects. For example:

```
rm(Hypertdf, Hypert2df)
```

Click **ALT> + Left Arrow** <- to return

To save the Environment to RData file
Ensure that you have your working directory set to Rstud_master
In the top left Environment panel click on the floppy disk icon
Enter the name WBenv2 into the dialogue box
Click save

Click **ALT> + Left Arrow** <- to return

$ naming convention
The data from individual columns in a data.frame can be addressed by using this convention. Thus for the data.frame WBdata we can specify any individual column using data.frame name + $ + column name. For example:

```
WBdata$Age
```

The function **str()** usefully gives all of the column names and data types in a data.frame. Each of the column names is prefixed with $:

```
str(WBdata)
```

Click **ALT> + Left Arrow** <- to return

Getting help
In either of the source or console panels

- Place the cursor over the function name and press <Fn 1>
- Type ? followed by function name and then run
- Use Search engine and enter 'R' followed by function name or question

Click **ALT> + Left Arrow** <- to return

Row column indexing
For detailed explanation of this please go to module no 5 “Data.frames indexing and subsetting”
Row / column indexing involves the use of square brackets and comma [,]

- Any expression to the **left** of the comma will be evaluated TRUE or FALSE for each row. That row will be included where the expression evaluates as TRUE
- Any expression to the **right** of the comma will be evaluated TRUE or FALSE for each column. That column will be included where the expression evaluates as TRUE

Click **ALT> + Left Arrow** <- to return

PlotPrevAsp() arguments

popdf	the original sample data.frame
testdf	the subset sample data.frame
AgeCol	name of agebands column (in quotes)
GenderCol	name of the gender column (in quotes)
mtitle	Main title for the graph

Click **ALT> + Left Arrow** <- to return