**Introduction to R**

Thanks to the following whose material have been used in creating this guide:

* Bruno Rodrigues, University of Strasbourg (Introduction to Programming: Econometrics with R)
* Rob Kabacoff ('Quick R' - <http://www.statmethods.net/index.html>)

1.1 Introduction

R is a modern implementation of the S language. S was developed at Bell Labs where the UNIX operating system, the C language as well as the C++ language were developed. As such R and S are very similar, but R is much more widely used, not least due to its free license. This allows users to freely share their own modifications of the software, thus allowing the widespread use of R worldwide.

A lot of pre-programmed routines are included in R, and you can add more through packages. One characteristic that’s important to recognise is that just as there are many ways of getting from Victoria Station to 102 Petty France, there are many ways of doing the same thing in R. Some ways are (computationally) faster, some are simpler to program, and some may be more conducive to your taste.

1.2 RStudio & coding

In Rstudio, there is a pane that is the command prompt, called the console. You can write commands directly in the console. For instance if you type:

x <- 3

(and press return) R creates an object called x which takes the value 3. You can see this in the ‘workplace’ (the environment window in the top right) and if you type:

x

the results are then shown in the command prompt. Results always appear in the console.

Note - to assign a name to an object in R you need to use an arrow and a hyphen:

<-

You can create a source code file in which you write the instructions by clicking file, new file, R Script. In R, these files have a .R extension. The code file should appear in the top left of the screen. These files can be saved and used again in future.

To execute the code, click run at the top of the screen or press ‘ctrl’ and ‘R’. R will run the line of code that the cursor is currently on, if you want to run several lines of code, highlight them and then press run (or ctrl and R).

On the right hand side of the screen, you can also see your files, any plots, packages (more about these shortly), and the help facility.

1.3 How to ask for help

1. To get information about a function e.g. the mean then type: help(mean).
2. If you get an error message, read the help file and if you still have trouble, try to read the error messages and understand them. For instance, if you enter the command:

> mean(y)

R could return the following error: *Error in mean(y) : object 'y' not found* and you need to understand what this means: here, you tried to calculate the mean without telling R what the variable y is.

1. You can use google to try and find the solution to the problem.

1.4 Exercises

1. Go to ‘File’, ‘New File’ and ‘R Script’ to open a new source code file in which you can store all commands you make during this exercise. Save it as ‘Into\_R\_Exercises.R’.
2. What does the command *head* do?
3. What command might you use to subset a dataset?

2.1 Setting up a working directory

It can be helpful to set up a working directory so that everything we are going to import into R or export from R will be saved by default into this folder. We can use the setwd() command (which stands for set working directory):

> setwd("S:/HQ/102PF/Shared/CJG\_OMS/OMS/Analytical Services/RACC/Statistical Methods and Development/4. Training and development/R training/Introduction")

You can check what the working directory currently is by using the getwd() command:

> getwd()

2.2 Importing data

You can import data in .csv files using Rstudio by clicking on the *Environment* tab and then the *Import Dataset* button. You can then navigate to the folder where the dataset "Offenders\_Chicago\_Police \_Dept.csv" is saved and click on it. A window will then appear which will include on the bottom right a preview of your data. Here it looks good, so we can click on import.

You can now see by looking in the ‘environment’ window that an object has been created (the ‘offenders’ dataset), and that it has 1413 observations and 9 variables.

Now look at the Console tab. You should see the command read.csv() appear with the whole path to the data set. It is a good idea to copy and paste this inside your script, so you won't need to do this again to load the data.

Alternatively you can simply use the command read.csv():

> offenders <- read.csv("Offenders\_Chicago\_Police\_Dept\_Main.csv")

This assumes by default that the first line of the file contains a header (header = T) and the columns are separated by a comma symbol (sep = ",").

There are other commands and various packages that can be used to import datasets with other extensions (e.g. .xls) e.g. see <http://www.statmethods.net/input/importingdata.html> .

2.3 Packages

Packages extend R's functionality enormously and are a key factor in making R so popular. For instance, to install the foreign package in R, use the *Install* button from the Packages tab in Rstudio.

This uses the following command:

> install.packages("foreign")

Once a package is *installed*, you should be able to see it in the packages tab. If you want to use it, you can *load* it by ticking the appropriate box in the packages window. You’ll see that the command to load the foreign package is:

> library("foreign")

To know more about a package, it is always useful to read the associated documentation:

> help(package=foreign)

2.4 Viewing & summarizing the dataset

As noted in the previous section, you can see by looking in the ‘environment’ window that the ‘offenders’ dataset has 1413 observations and 9 variables. To view this dataset, click the icon to the right of this information, which you can see from the console is the equivalent of typing the following command:

> View(offenders)

So we can see for the first observation that the surname is Lightning, the firstname Dwayne, the block where he is supposed to reside 0000X E 100TH ST, his gender male, his race black, his birthday 26 May 1988, his height 6 foot 4 inches, his weight 215 lbs and his victim is a minor.

To obtain a summary of the meta-data of your dataset you can click on the arrow by 'offenders' in the 'environment' window, which provides the same information as by typing the following command:

> str(offenders)

Looking at the output provided informs you that the dataset offenders is in R terminology a dataframe, and as we’ve already seen has 1413 observations and 9 variables. Also provided is some information about each variable in the dataset (or dataframe) as designated by R; the name, the type (in this case either integer or factor, the latter assuming the variable is a categorical one with different levels e.g. gender with female =1 and male =2), and the first 10 observations.

If you want to view only the top 10 rows of the dataset:

> head(offenders,10)

If you want to view only the last 2 rows of the dataset:

> tail(offenders, 2)

Square brackets can be used to subset data. For instance

>offenders[ i , j ]

would return the value in the ith row and jth column of the dataframe offenders. Use a small c to concatenate

If you want the fourth and fifth variables for the 500th and 502nd observation:

> offenders[c(500, 502),4:5]

Note the use of square brackets to specify which rows and columns you want, the rows being specified first and then the columns, the two being separated by a comma. c stands for concatenate, enabling us to retrieve the 500th and 502nd observations together.

Another very useful and complementary summary command is:

> summary(offenders)

This provides the min, max, quartiles, median and mean for each numeric or integer variable. For factor variables, the number of observations is shown together with the names of the first six levels.

To ascertain a particular summary statistic (e.g. a standard deviation) for each of the variables that is not provided by the summary command, you can also use the sapply command:

> sapply(offenders, sd)

If you then want the mean of WEIGHT by GENDER, you can also use the tapply command:

> tapply(offenders$WEIGHT, offenders$GENDER, mean)

2.5 Using dataset variables

To view a specific variable, for instance gender, you use a dollar sign as follows:

> offenders$GENDER

The format is *dataframe name, $, variable name*. An alternative way would be:

> offenders[,4]

To create a new variable providing the weight in kg:

> offenders$weight\_kg <- offenders$WEIGHT\*0.45359

You can view the new variable by using the View command (note that after make changes to the dataset you need to use the view command again to see the latest version) and summarize it using the summary command (notice the maximum weight is rather high – we can explore that later) and use the str or class command to see that it’s of type numeric:

> class(offenders$weight\_kg)

So far, our dataset offenders has variables of three different classes; integer, number, and factor. Other useful types are character, logical and date, which we'll look at shortly.

2.6 Exercises

1. Open source code file ‘Into\_R\_Exercises.R’ and store all commands you make during this exercise in it.
2. Call up the package *dplyr*
3. Read in dataset Offenders\_Chicago\_Police\_Dept\_Age.csv, calling it offenders\_age.
4. Read in dataset Offenders\_Chicago\_Police \_Dept\_Trial.csv, calling it offenders\_trial.
5. What are the mean, max and min for variable AGE in the dataset offenders\_age?
6. What is the median for variable AGE in the dataset offenders\_age when only considering the observations 50 to 100?
7. Create a new variable providing the number of (full) years over 16.

3.1 Dates

As you'll maybe have noted, BIRTH\_DATE in the ‘offenders’ dataset is presently a factor variable. To understand the variable better we will want to change it to have class date.

Class date involves dates being represented in R as the number of days since 1970-01-01, with negative values for earlier dates. The format is year (4 digits) - month (2 digits) - day (2 digits). You can see this if we ask R for today's date:

> Sys.Date()

To change BIRTH\_DATE to have class date we need to use the as.Date function which allows us to tell R how to interpret the date format we have using the following symbols:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Meaning** | **Example** |
| **%d** | day as a number (0-31) | 01-31 |
| **%a %A** | abbreviated weekday  unabbreviated weekday | Mon Monday |
| **%m** | month (00-12) | 00-12 |
| **%b %B** | abbreviated month unabbreviated month | Jan January |
| **%y %Y** | 2-digit year  4-digit year | 07 2007 |

Source: 'Quick R' - <http://www.statmethods.net/input/dates.html>

If we look at BIRTH\_DATE (the first observation is 05/26/1988) we can see that the current format is two digit month represented by %m followed by / followed by the two digit day represented by %d followed by / and ending with the four digit year represented by %Y. We can therefore make a new date variable (called birth\_date\_formatted) with class date as follows:

> offenders$birth\_date\_formatted <- as.Date(offenders$BIRTH\_DATE, "%m/%d/%Y")

Now we have a variable with class date we can create new variables containing just part of the date e.g.

> offenders$b\_wkday <- weekdays(offenders$birth\_date\_formatted)

> offenders$b\_qtr <- quarters(offenders$birth\_date\_formatted)

And using a package called lubridate:

> offenders$b\_year <- year(offenders$birth\_date\_formatted)

> offenders$b\_month <- month(offenders$birth\_date\_formatted)

> offenders$b\_day <- day(offenders$birth\_date\_formatted)

You can also calculate the number of days since a date. For instance let's say we want to know the no. of days between the date of birth and 1 Jan 2000:

> offenders$days\_before\_01\_01\_2000 <- as.Date("2000-01-01") - offenders$birth\_date\_formatted

3.2 Exercises

1. Open source code file ‘Into\_R\_Exercises.R’ and store all commands you make during this exercise in it.
2. Read in dataset 'FTSE\_12\_14.csv' and change the date variable to have class date.
3. Calculate a weekday variable (called b\_wkday) and a daily performance (close price minus open price) variable (called performance).
4. See which weekday performance is best using the function tapply(FTSE$performance, FTSE$b\_wkday, mean)?

4.1 Reordering the factor class, coercing classes, the logical class and Ifelse

As mentioned previously, factors are for categorical variables involving different levels. So for example, in the dataset offenders, FEMALE is stored as 1, and MALE as 2. We can see this from looking at the environment tab (after clicking the arrow to the left of offenders) and also from using the following command:

> levels(offenders$GENDER)

The ordering is useful when we do regression analyses as we may want a particular category to be the reference category. By default, the first category is the reference category but this can be changed e.g. from FEMALE to MALE using the following command:

> offenders$GENDER <- relevel(offenders$GENDER, "MALE")

As we've demonstrated already using as.Date, it's possible to coerce variables from one class to another. We can change the HEIGHT variable in the offenders dataset to be a numeric variable as follows:

> offenders$HEIGHT <- as.numeric(offenders$HEIGHT)

and back again as follows:

> offenders$HEIGHT <- as.integer(offenders$HEIGHT)

We can change the BLOCK variable in the offenders dataset to be a character variable as follows:

> offenders$BLOCK <- as.character(offenders$BLOCK)

and back again as follows:

> offenders$BLOCK <- as.factor(offenders$BLOCK)

A logical variable has two values TRUE or FALSE. If you type the following:

> y <- 4 > 3

(asking if 4 is greater than 3)

then R returns that l is TRUE and we can see l is of class logical:

> class(y)

Another example, adding a logical variable to a data frame, using the FTSE data:

> FTSE$price\_rise <- FTSE$Close – FTSE$Open

The logical class is important to perform the ifelse routine. We can use this routine, for example, to create a new variable identifying those with height over 6 feet in the dataset offenders:

> offenders$height\_over\_6feet <- ifelse(offenders$HEIGHT>=600, 1, 0)

4.2 Conditional summary statistics & further statistical functions

We’ve already covered obtaining summary statistics for all observations and also selecting certain observations according to row number and column number. So if you want summary statistics for the 50th to 500th and 502nd observations of the fourth and fifth variables:

> summary(offenders[c(50:500, 502),4:5])

Let's say you are only interested in having a summary of height for those of white ethnicity:

> summary(offenders$HEIGHT[offenders$RACE=="WHITE"])

Or you are interested in having a summary of height for those with weight over 200lbs:

> summary(offenders$HEIGHT[offenders$WEIGHT>200])

Another useful function to know is table. This gives the frequencies of a variable:

> table(offenders$RACE)

To get the relative frequencies, you can divide by the number of observations:

> table(offenders$RACE)/length(offenders$RACE)

Alternatively, give the frequency table a name and then use prop.table:

>race\_tab <- table(offenders$RACE)

>prop.table(race\_tab)

To convert this into a percentage frequency table:

>round(prop.table(race\_tab)\*100,1)

Two way frequency tables can be created by simply adding another term:

>table(offenders$RACE, offenders$GENDER)

Three way tables can be created using the function ‘ftable’.

Other useful numeric and statistical functions include:

* abs(x): returns the absolute value of x
* sqrt(x): returns the square root of x
* round(x, digits = n): rounds a number to the nth place
* exp(x): returns the exponential of x
* log(x): returns the natural log of x
* sum(x): if x is a vector, returns the sum of its elements
* min(x): if x is a vector, returns the smallest of its elements
* max(x): if x is a vector, returns the biggest of its elements
* rnorm(n, mean = 0, sd = 1): return n random numbers from the standard normal distribution
* rbinom(n, no. of trials = 1, prob = 0.5): return random numbers from n coin tosses
* mean(x): if x is a vector of observations, return the mean of its elements
* sd(x): if x is a vector of observations, return its standard deviation
* cor(x): gives the linear correlation coefficient
* median(x): if x is a vector of observations, return its median

4.3 Exercises

1. Open source code file ‘Into\_R\_Exercises.R’ and store all commands you make during this exercise in it.
2. Using the dataset offenders (Offenders\_Chicago\_Police\_Dept\_Main.csv) change the reference category of the RACE variable to BLACK.
3. Create a new variable called 'height\_under\_4feet' which is 1 if under 4 feet and 0 otherwise.
4. How many have heights of less than 4 feet, what are their (recorded) heights and gender(s)?
5. Produce a table showing the counts of HEIGHT including missing values.
6. Calculate a trimmed mean (excluding 5% of observations either side) for weight.

5.1 Merging datasets

Let’s merge the datasets offenders and offenders\_age using the combination of fields that together form a unique identifier.

> offenders\_merge <- merge(offenders, offenders\_age, by=c(“LAST”, “BIRTH\_DATE”, “VICTIM\_MINOR”))

For more information about the different sorts of joins see: <http://www.dummies.com/programming/r/how-to-use-the-merge-function-with-data-sets-in-r/>

We can also join two datasets vertically or horizontally, using the rbind or cbind functions respectively. First, let’s create some new datasets.

> height1 <- offenders\_merge[1:10, “HEIGHT”]

> height2 <- offenders\_merge[11:20, “HEIGHT”]

> weight1 <- offenders\_merge[1:10, “WEIGHT”]

We can use rbind() to join vertically height1 and height2:

> height <- rbind(height1, height2)

Note that the two data frames must have the same variables, although they do not have to be in the same order.

Now use cbind() to join horizontally height1 and weight1:

> height\_weight <- cbind(height1, weight1)

5.2 Handling missing values

In R, missing values are represented by the symbol NA (not available). Impossible values (e.g. dividing by zero) are represented by the symbol NaN (not a number).

We can easily make a logical vector with value TRUE if the corresponding element in is NA, and FALSE otherwise:

> Miss <- is.na(offenders\_merge$HEIGHT)

Then we can use this to ascertain the number of missing values the height variable contains:

> length(Miss[Miss == TRUE])

We can also create a logical vector showing whether the row is complete (TRUE) or has a missing value in one or more columns (FALSE):

> complete.cases(offenders\_merge)

Then we can use this to create a list of cases which have missing values:

> offenders\_merge\_missings <- offenders\_merge[!complete.cases(offenders\_merge),]

We can also create a list of cases which don’t have missing values:

> offenders\_merge\_completes <- offenders\_merge[complete.cases(offenders\_merge),]

5.3 Exporting data

> write.csv(offenders\_merge, file = "Offenders\_Chicago\_Police\_Dept\_.csv")

This assumes by default that you want to export the row headers and that the columns are separated by a comma symbol (sep = ",").

5.4 Exercises

1. Open source code file ‘Into\_R\_Exercises.R’ and store all commands you make during this exercise in it.
2. Merge offenders\_trial with offenders\_age, creating a new dataset called offenders\_trial\_age.
3. Which age group has received the most trials?
4. Export the dataset offenders\_trial\_age to a csv file.
5. Using offenders create a new variable HEIGHT\_NEW which is as HEIGHT except with the missing values replaced by the average height.
6. Produce a table showing the counts of HEIGHT including missing values.