Intro R training

## 1. Introduction to R

### Session aims

* Overview of R
* Understand how it’s used in MoJ
* Demonstrate its basic functions

### What is R?

R is an open-source programming language and software environment, designed primarily for statistical computing. It has grown rapidly in popularity in the last few years due to increased interest in data science.

Some of the advantages:

* It is **popular** - there is a large, active and rapidly growing community of R programmers, which has resulted in a plethora of resources and extensions.
* It is **powerful** - the history as a statistical language means it is well suited for data analysis and manipulation.
* It is **extensible** - there are a vast array of **packages** that can be added to extend the functionality of R, produced by statisticians and programmers around the world. These can range from obscure statistical techniques to tools for making interactive charts
* It’s **free and open source** - a large part of its popularity can be owed to its low cost, particularly relative to proprietary software such as SAS.

### How is it used?

There a number of areas in the MoJ where R is making an impact:

1. The Reproducible Analytical Pipeline (RAP) is a set of tools and standards for producing our statistical publications in a more automated and reproducible way. The Offender Management Statistics Quarterly publication already runs on RAP <https://github.com/moj-analytical-services/OMSQ_RAP>
2. A number of webapps using Shiny have been produced, allowing customers to explore data in an interactive way. The PQ tool is a strong example: <https://pq-tool.apps.alpha.mojanalytics.xyz/>

### Contents

1. Basic set up - in this section you will be introduced to the RStudio Environment and some of the basic R commands
2. Processing data - this section involves importing data into R and inspecting it
3. Calculations - this section introduces some basic functions for data manipulation
4. Dates - this section explores working with dates
5. Merging data, missing values and exporting - This section explores some more advanced data wrangling techniques, as well as exporting data.

### Basic setup

Prior to joining the session, you should ensure you are set up on the Analytical Platform (IT system permitting). <https://moj-analytical-services.github.io/platform_user_guidance/getting-started.html>.

This guide is hosted on Github: <https://github.com/moj-analytical-services/intrortraining>

In that repository, there is the file code\_participant.R, which contains code related to the exercises in this guide. The Data Science team conduct separate training sessions for working for github, but for now, you will just want to import the project from github by following the steps outlined in section 4.1.3 of the platform guidance: <https://moj-analytical-services.github.io/platform_user_guidance/using-github-with-r-studio.html>

### 1.1 Command console

The window in which commands are entered is called the “console” window. It is used to input and execute code. Results, errors and warnings are shown directly in the same window.

A command which is entered into the console is executed by simply pressing enter.

Code appears in blue text. Results appear in black text. Warnings and errors appear as red text.

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

## [1] 3

The results are then shown in the console.

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

Furthermore, R is case sensitive so if you were to type X, an error would be displayed as you have not yet created and object called X.

The console will remember your most recent commands, if you want to reuse one, just use the up and down arrows to scroll through them. When you have found the one that you want, press return, R will repeat that line of code and display the results.

### 1.2 Script file

While commands can be written into the console, it is a good idea to use a “script”. This is a code file that can be saved and reused in future.

To create a new file: click ‘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. 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.

### 1.3 Other windows and getting help

The top right panel shows all “objects” that are in your working environment. This will become clearer throughout the session but typically, this will be any data that you have created or imported, additional variables and values that you have created. For instance, if you have run the code above, ‘x’ should be shown here. Other objects such as regression models that you have created would also appear here. From here you can also use a drop down menu to import more data.

The bottom right window has several tabs. You can see your files and any plots that you have created. It also shows what packages are available and what ones are loaded, more on packages later.

There is also a help menu. You can either use this or type ? into the console and then the name of what it is you want help on in brackets. For instance, the following line would give you help on the function called ‘mean’:

?mean

Of course, you can also use google or the ASD slack to try and find the solution to the problem.

### 1.4 Exercises

1. Create a new source code file in which you can store all commands you make during this exercise. Save it as ‘Intro\_R\_Exercises.R’.
2. Create a new value called y which is equal to 17.
3. Now multiply y by 78. What answer do you get?
4. What does the command head do?
5. What command might you use to subset a dataset?

## 2. Processing data

### 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).

Note, you won’t need to do this if you are working on the analytical platform but it’s still a very useful command to know (note it’s commented out using the # command as the path is only for DOM1):

# setwd("S:/HQ/102PF/Shared/CJG\_OMS/OMS/Analytical Services/RACC/Statistical Project Delivery/3. Training and development/R training/Introduction")

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

getwd()

### 2.2 Packages

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

This uses the following command:

install.packages("tidyverse")

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 can also load packages using the library command, which you can inside your script, so they will automatically load when you run it.:

library("tidyverse")

FALSE ── Attaching packages ────────────────────────────────────────────────────────────────────────────────── tidyverse 1.2.1 ──

FALSE ✔ ggplot2 2.2.1.9000 ✔ purrr 0.2.4   
FALSE ✔ tibble 1.3.4 ✔ dplyr 0.7.4   
FALSE ✔ tidyr 0.7.2 ✔ stringr 1.3.0   
FALSE ✔ readr 1.1.1 ✔ forcats 0.2.0

FALSE ── Conflicts ───────────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──  
FALSE ✖ dplyr::filter() masks stats::filter()  
FALSE ✖ dplyr::lag() masks stats::lag()  
FALSE ✖ dplyr::vars() masks ggplot2::vars()

The package tidyverse contains many useful packages such as dplyr which is a particularly useful package for manipulating and processing data. Many of the functions in the rest of this training course are from this package.

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

help(package=dplyr)

### 2.3 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\_Main.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 commands library and read\_csv() appear with the whole path to the data set. It is a good idea to copy and paste these commands inside your script, so you won’t need to do this again to load the data.

Alternatively (you have already loaded the readr package as it is part of the tidyverse package - see section 2.2) you can simply use the command read\_csv():

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

## Parsed with column specification:  
## cols(  
## LAST = col\_character(),  
## FIRST = col\_character(),  
## BLOCK = col\_character(),  
## GENDER = col\_character(),  
## REGION = col\_character(),  
## BIRTH\_DATE = col\_character(),  
## HEIGHT = col\_integer(),  
## WEIGHT = col\_integer(),  
## PREV\_CONVICTIONS = col\_double(),  
## SENTENCE = col\_character(),  
## AGE = col\_integer()  
## )

Note that the above assumes that the csv file is in your working directory, otherwise you will need to include the file path - see section 2.1.

#### Importing data from the analytical platform amazon server

To import the data for this session from the platform amazon server use this command:

offenders <- s3tools::s3\_path\_to\_full\_df("alpha-everyone/R\_training\_intro/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>

### 2.4 Inspecting 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 using the command:

View(offenders)

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)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 1413 obs. of 11 variables:  
## $ LAST : chr "RODRIGUEZ" "MARTINEZ" "GARCIA" "RODRIGUEZ" ...  
## $ FIRST : chr "JUAN" "MOISES" "ELLIOTT" "JOSE" ...  
## $ BLOCK : chr "009XX W CUYLER AVE" "011XX N KILBOURN AVE" "011XX W 18TH ST" "012XX W RACE AVE" ...  
## $ GENDER : chr "MALE" "MALE" "MALE" "MALE" ...  
## $ REGION : chr "West" "East" "South" "North" ...  
## $ BIRTH\_DATE : chr "06/22/1955" "02/07/1954" "08/11/1970" "02/10/1959" ...  
## $ HEIGHT : int 198 198 201 237 201 199 201 236 198 199 ...  
## $ WEIGHT : int 190 180 200 195 220 130 200 235 140 130 ...  
## $ PREV\_CONVICTIONS: num 0 0 0 0 0 0 0 0 0 0 ...  
## $ SENTENCE : chr "Court\_order" "Prison\_<12m" "Court\_order" "Court\_order" ...  
## $ AGE : int 58 59 43 54 34 50 32 43 34 18 ...  
## - attr(\*, "spec")=List of 2  
## ..$ cols :List of 11  
## .. ..$ LAST : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ FIRST : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ BLOCK : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ GENDER : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ REGION : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ BIRTH\_DATE : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ HEIGHT : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_integer" "collector"  
## .. ..$ WEIGHT : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_integer" "collector"  
## .. ..$ PREV\_CONVICTIONS: list()  
## .. .. ..- attr(\*, "class")= chr "collector\_double" "collector"  
## .. ..$ SENTENCE : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ AGE : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_integer" "collector"  
## ..$ default: list()  
## .. ..- attr(\*, "class")= chr "collector\_guess" "collector"  
## ..- attr(\*, "class")= chr "col\_spec"

Looking at the output provided informs you that the dataset ‘offenders’ is in R terminology both a tibble and 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, number or character).

The summary command also provides some useful details:

summary(offenders)

## LAST FIRST BLOCK   
## Length:1413 Length:1413 Length:1413   
## Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character   
##   
##   
##   
##   
## GENDER REGION BIRTH\_DATE HEIGHT   
## Length:1413 Length:1413 Length:1413 Min. : 24.0   
## Class :character Class :character Class :character 1st Qu.:200.0   
## Mode :character Mode :character Mode :character Median :200.0   
## Mean :209.1   
## 3rd Qu.:236.0   
## Max. :239.0   
## NA's :24   
## WEIGHT PREV\_CONVICTIONS SENTENCE AGE   
## Min. :100.0 Min. : 0.00 Length:1413 Min. :15.00   
## 1st Qu.:160.0 1st Qu.: 1.40 Class :character 1st Qu.:34.00   
## Median :180.0 Median : 9.80 Mode :character Median :43.00   
## Mean :188.2 Mean :12.83 Mean :43.88   
## 3rd Qu.:210.0 3rd Qu.:22.40 3rd Qu.:52.00   
## Max. :601.0 Max. :39.20 Max. :94.00   
##

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’. so, if you want the fourth and fifth variables for the 500th and 502nd observation:

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

## # A tibble: 2 x 2  
## GENDER REGION  
## <chr> <chr>  
## 1 MALE West  
## 2 MALE South

Note that c stands for concatenate, enabling us to retrieve the 500th and 502nd observations together. The colon enables us to retrieve from the 4th to the 5th columns.

### 2.5 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]

## # A tibble: 1,413 x 1  
## GENDER  
## <chr>  
## 1 MALE  
## 2 MALE  
## 3 MALE  
## 4 MALE  
## 5 MALE  
## 6 MALE  
## 7 MALE  
## 8 MALE  
## 9 MALE  
## 10 MALE  
## # ... with 1,403 more rows

We can also use the select() command from the dplyr package to choose just the variables from the ‘offenders’ dataset that we want:

select(offenders, LAST, FIRST)

Note (as is common with dplyr functions) we first specify the name of the dataset and then the variables that we want.

To create a new variable providing the weight in kg (instead of in pounds):

offenders$weight\_kg <- offenders$WEIGHT\*0.454

You can alternatively use the command mutate() which is part of the dplyr package.

?mutate

So to create weight\_kg using mutate:

offenders <- mutate(offenders, weight\_kg = WEIGHT \* 0.454)

As we have called our variable “weight\_kg”, R has overwritten the weight\_kg variable we have already created.

You can see 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):

View(offenders)

We can rename variables using the dplyr function rename(). Let’s change the name of BIRTH\_DATE in our ‘offenders’ dataset.

offenders <- rename(offenders, DoB = BIRTH\_DATE)

### 2.6 Exercises

1. What are the mean, max and min for variable AGE in the dataset offenders?
2. Create a new dataset creating a new variable providing the number of (full) years someone is over 16 and drop the orginal WEIGHT variable.

## 3 Calculations

### 3.1 Data classes

All variables have an associated class. The class will determine what calculations are possible with them and how R should treat them. So far, our dataset offenders has variables of three different classes; integer, number, and character. Other useful types are factor, logical and date.

We can check what class a variable is using summary, looking at the information in the Environment pane or by using the command “class”

class(offenders$weight)

## Warning: Unknown or uninitialised column: 'weight'.

## [1] "NULL"

It’s possible to coerce variables from one class to another. We can change the weight variable in the offenders dataset to be a numeric variable as follows:

offenders$WEIGHT <- as.numeric(offenders$WEIGHT)

and back again as follows:

offenders$WEIGHT <- as.integer(offenders$WEIGHT)

We can change the GENDER variable in the offenders dataset to be a factor variable as follows:

offenders$GENDER <- as.factor(offenders$GENDER)

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 now when looking at the environment tab (after clicking the arrow to the left of offenders) and also the order from using the following command:

levels(offenders$GENDER)

## [1] "FEMALE" "MALE"

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")

We can now change the GENDER variable in the offenders dataset back to be a character variable as follows:

offenders$GENDER <- as.character(offenders$GENDER)

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

offenders$tall <- offenders$HEIGHT > 175

Take care of the sytnax here, the first arrow and hyphen are used to assign the new variable. The second arrow is to compare height with 174. In this case we are asking if the offenders height is greater than 175cm.

We can see offenders$tall is of class logical:

class(offenders$tall)

## [1] "logical"

### 3.2 Ifelse

The logical class is important to perform ifelse commands. These are the equivalent of If statements in Excel. We can use this, for example, to identify those with weight under 90kg in the dataset ‘offenders’:

offenders$wt\_under\_90 <- ifelse(offenders$weight\_kg<90, 1, 0)

### 3.3 Grouping and summarising data

We can produce breakdowns of statistics using the group\_by and summarise commands from the dplyr package.

?dplyr::summarise  
?group\_by

group\_by() identifies which variables we want to produce breakdowns by.

summarise() is used to indicate which values we want to calculate.

Using these functions together we can replicate the apply family of functions in base R and produce summary statistics in a similar way to pivot tables in Excel. We can use these functions together using a %>% operator.

So if we want the mean number of previous convictions with breakdown by REGION and GENDER:

regional\_gender\_average <- offenders %>% group\_by(REGION, GENDER) %>%  
summarise(Ave = mean(PREV\_CONVICTIONS))

In the above code R is taking the ‘offenders’ dataset, grouping it first by REGION and then by GENDER (note that the dataset being passed into the group\_by command using ‘%>%’ is instead of specifying the ‘offenders’ dataset as the first argument within the group\_by command), and then outputting the mean number of previous convictions by REGION and GENDER. The new mean number of previous convictions variable is called ‘Ave’. The results are saved into a new dataset called ‘regional\_gender\_average’.  
There are other functions that could be used here instead of mean e.g. n, n\_distinct, min, max, mean, median, var and sd

If we want also to add a new variable called ‘Count’ that provides the counts by REGION and GENDER we can rerun as follows:

regional\_gender\_average <- offenders %>% group\_by(REGION, GENDER) %>%  
summarise(Ave = mean(PREV\_CONVICTIONS), Count=n())

The regional\_gender\_average dataset is grouped by REGION and GENDER. If we want to remove this grouping we can use the ungroup command:

regional\_gender\_average <- ungroup(regional\_gender\_average)

### 3.4 Filter

We have selected a subset of columns or variables using select. Now let’s select a subset of rows or observations. A good function for that is filter() from the dplyr package.

Let’s first take a look at the different possible values of the SENTENCE variable. We can do that quickly using the group\_by/summarise combination.

offenders %>% group\_by(SENTENCE) %>% summarise(Count = n())

## # A tibble: 3 x 2  
## SENTENCE Count  
## <chr> <int>  
## 1 Court\_order 856  
## 2 Prison\_<12m 417  
## 3 Prison\_12m+ 140

To filter we just specify the data that we want to filter (offenders) and the value that we want to filter on. In this case lets filter where SENTENCE is “Court\_order” and AGE is more than 50 and then recalculate the mean number of previous convictions with breakdown by REGION and GENDER:

crt\_order\_average <- offenders %>% filter(SENTENCE == "Court\_order" & AGE > 50) %>% group\_by(REGION, GENDER) %>%  
summarise(Ave = mean(PREV\_CONVICTIONS))

### 3.5 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

### 3.6 Exercises

1. Using group\_by and summarise, calculate the average and median age for females in the West.
2. Create a new variable called ‘height\_under\_150’ which is 1 if under 150 cm and 0 otherwise.
3. How many have heights of less than 4 feet, what are their (recorded) heights and gender(s)?
4. Produce a table showing the counts of height including missing values.

## 4 Dates

### 4.1 Manipulating dates

As you may have noted, DoB 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()

## [1] "2018-07-19"

To change DoB 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 | 31 |
| %a | abbreviated weekday | Mon |
| %A | unabbreviated weekday | Monday |
| %m | month(00 - 12) | 04 |
| %b | abbreviated month | Jan |
| %B | unabbreviated month | January |
| %y | 2 digit year | 18 |
| %Y | 4 digit year | 2018 |

Source: ‘Quick R’ - <http://www.statmethods.net/input/dates.html> If we look at DoB (the first observation is 05/26/1988) we can see that the current format is two digit month represented by %m, followed by a /, 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 DoB\_formatted) with class date as follows:

offenders$DoB\_formatted <- as.Date(offenders$DoB, "%m/%d/%Y")

The expression %m/%d/%Y tells R what the data currently looks like so it knows where to find the day, month and year needed to create a date.

Remember, to refer to a specific variable, we use a dollar sign. We have used a variable name that doesn’t currently exist in ‘offenders’ so R has created a new variable and appended it.

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

offenders$b\_wkday <- weekdays(offenders$DoB\_formatted)

offenders$b\_qtr <- quarters(offenders$DoB\_formatted)

And using a package called lubridate:

library(lubridate)  
offenders$b\_year <- year(offenders$DoB\_formatted)  
offenders$b\_month <- month(offenders$DoB\_formatted)  
offenders$b\_day <- day(offenders$DoB\_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$days\_before\_2000 <- as.Date("2000-01-01") - offenders$DoB\_formatted

### 4.2 Exercises

1. Read in dataset ‘FTSE\_12\_14.csv’ and change the date variable to have class date. (s3tools::s3\_path\_to\_full\_df(“alpha-everyone/R\_training\_intro/FTSE\_12\_14.csv”))
2. Calculate a weekday variable and another variable to measure daily performance, that is, how much the share price increases or decreases on a daily basis (close price minus open price).
3. See which weekday performance is best using summarise or creating a table.

## 5 Merging data, missing values and exporting

### 5.1 Merging datasets

There are dplyr functions left\_join, right\_join, inner\_join, full\_join, semi\_join and anti\_join which can merge data sets, provided you have some common fields to match on. This is similar to SQL.

Let’s import a new dataset which contains informaiton on whether the offenders faced trial.

offenders\_trial <- read\_csv("Offenders\_Chicago\_Police\_Dept\_Trial.csv")

## Parsed with column specification:  
## cols(  
## LAST = col\_character(),  
## DoB = col\_character(),  
## TRIAL = col\_character()  
## )

Or using the platform:

offenders\_trial <- s3tools::s3\_path\_to\_full\_df("alpha-everyone/R\_training\_intro/Offenders\_Chicago\_Police\_Dept\_Trial.csv")

We merge the datasets with offenders using the combination of fields that together form a unique identifier.

offenders\_merge <- dplyr::inner\_join(offenders, offenders\_trial, by=c("LAST", "DoB"))

For more information about the different sorts of joins and other data transformation functions see the ‘Data Transformation Cheat Sheet’ at: <https://www.rstudio.com/resources/cheatsheets/>

We can also join two datasets vertically or horizontally, using the bind\_rows() or bind\_cols() functions respectively.

If we have two datasets with the same variables, we can use bind\_rows to join them vertically.

For instance:

men <- filter(offenders, GENDER == "MALE")  
women <- filter(offenders, GENDER == "FEMALE")  
rejoined <- rbind(men, women)

Note that ‘rejoined’ has the same number of observations and variables as ‘offenders’.

The bind\_cols function does something similar but appends data horizontally. Be sure the rows align before using this function!

### 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 look at the HEIGHT variable as previously:

height\_table <- offenders %>% group\_by(HEIGHT) %>% summarise(Count=n())

Then we can view the height\_table we’ve made which will include the number of missing values the height variable contains:

View(height\_table)

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)

Using filter we can create a new data frame just with those that are complete:

complete\_offenders <- filter(offenders, complete.cases(offenders))

### 5.3 Exporting data

A command to export data into csv format is write.csv. For instance, to export our data which contains the complete cases:

write.csv(complete\_offenders, file = "Complete\_offenders.csv")

This assumes by default that you want to export the row headers and that the columns are separated by a comma symbol (sep = “,”). The data will be saved as a CSV in your working directory.

### 5.4 Exercises

1. Creating a new dataset called offenders\_trial\_age which includes the data in offenders\_trial and the age column of offenders.
2. Export the dataset offenders\_trial\_age to a csv file.
3. Using offenders create a new variable HEIGHT\_NEW which is as HEIGHT except with the missing values replaced by the average height (hint: use the is.na() function).