Intro R training

## Basic setup

This document provides accompanying training material used in the Introduction to R Training session, conducted by the Data & Analysis R training group. Prior to joining the session, you should ensure you are set up on the Analytical Platform - see <https://user-guidance.services.alpha.mojanalytics.xyz/introduction.html#get-started>.

You should then work through the following:

1. <https://user-guidance.services.alpha.mojanalytics.xyz/introduction.html#deploy-analytical-tools>
2. <https://user-guidance.services.alpha.mojanalytics.xyz/introduction.html#open-analytical-tools>
3. <https://user-guidance.services.alpha.mojanalytics.xyz/introduction.html#configure-git-and-github>
4. Clone the GitHub repository for this course (<https://github.com/moj-analytical-services/IntroRTraining>) by following step 1 here: <https://user-guidance.services.alpha.mojanalytics.xyz/github.html#r-studio>
5. After cloning the repository, open up the file called ‘code\_participant.R’ in RStudio, which contains example code that accompanies this course. To do this, look under the ‘Files’ tab in the bottom-right corner and click where it says ‘code\_participant.R’.
6. In the Console window in RStudio, enter this command to make sure you have the required packages installed: renv::restore()
7. If you haven’t already, request access to the alpha-r-training bucket on Amazon S3 (which is used to store some example data) from the session organisers. To check if you can access the bucket you can run the following code in the RStudio Console, which should output a list of files stored in the bucket: botor::s3\_ls('s3://alpha-r-training')

## Contents

1. Introduction - in this section you will be introduced to R, the RStudio Environment and some of the basic R commands
2. Processing data - this section introduces R packages, importing data into R, inspecting data and data classes
3. Data wrangling and ‘group by’ calculations - this section covers the most useful data manipulation functions using the dplyr package
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
6. Extra resources

## 1. Introduction

### 1.1 Session aims

* Overview of R
* Get new R users up and running with confidence
* Introduce key functions needed for Data & Analysis work

### 1.2 What is R?

R is an open-source programming language and software environment, designed primarily for statistical computing. It has a long history - it is based on the S language, which was developed in 1976 in Bell Labs, where the UNIX operating system and the C and C++ languages were developed. The R language itself was developed in the 1990s, with the first stable version release in 2000.

R has grown rapidly in popularity particularly in the last five years, due to the increased interest in the data science field. It is now a key tool in the MoJ’s Analytical Platform.

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.

### 1.3 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/>
3. It has enabled more technical analysis to be done with the help of packages written by academics and statisticians, which would have to be coded from scratch using SAS. For example, the PQ tool makes use of packages to facilitate Natural Language Processing and Text Mining.

### 1.4 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) the assignment operator (less than followed by hyphen) in R assigns the name x to the object taking 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 console.

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.

As well as storing single values, you can also create vectors. The below statement creates a vector object with the values 3, 2 and 4:

x <- c(3, 2, 4)

This uses the “c” function - the c is short for “concatenate”. You can now see the new object x in the ‘workplace’. Note that the old object x has been overwritten and that the new object is of class numeric.

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.

### 1.5 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.

### 1.6 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 containing for instance the details of 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 which packages are available and which 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 ASD slack to try and find the solution to the problem.

### 1.7 Exercises

1. Create a new R script 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?

## 2. Processing data

### 2.1 Setting up a working directory

The default behaviour of RStudio for the handling of files (e.g. datasets, code scripts, etc) is to use a working directory, which is a folder where RStudio reads and saves files. Therefore, before we start writing any code we should set up a working directory so that everything we are going to import into RStudio or export from RStudio will be saved by default into this folder.

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

getwd()

If you want to change the working directory you can use the setwd() command:

setwd("~")

If you want to change the working directory to a specific repository/folder you have cloned from GitHub, for example, the “IntroRTraining” repository, you can use the setwd() command as follows:

setwd("~/IntroRTraining")

Additionally, you can set your working directory manually following the steps below:

1. Create a folder with an appropriate name containing any files you need for your RStudio session.
2. From RStudio, use the menu to change your working directory under Session > Set Working Directory > Choose Directory.
3. Choose the directory (folder) you’ve just created in step 1.

If you have not synced your RStudio to GitHub or are unable to clone a repository from GitHub you can create a new project using RStudio and upload files from a folder in your laptop into this new project. You can then set your working directory to this new project that you have created. Steps to create a new project are detailed below:

1. Click on the File Menu and select New Project.
2. Select New Directory
3. Select New Project
4. Write an appropriate name for your project in the Directory Name text box and select Create Project.
5. Upload files to your new project by clicking Upload in the Files window menu (bottom right window) and select Browse to upload any files saved in a folder e.g. data csv files saved in a folder in your OneDrive.
6. Set your working directory to your new project using the RStudio commands above (or using the manual steps).

### 2.2 Packages

A lot of pre-programmed routines are included in R, and you can add a lot 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.

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, which we recommend you use for data manipulation, use the Install button from the Packages tab in Rstudio.

This uses the following command:

install.packages("tidyverse")

Note that if you are using R on the Analytical Platform the tidyverse package may already be installed, hence the above step can be skipped.

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 put inside your script, so they will automatically load when you run it:

library("tidyverse")

The package suite 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 introductory training are from this package.

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

?dplyr

### 2.3 Importing data

It is important to be able to import data both from your own computer as well as the Analytical Platform cloud storage.

#### Importing data using a local version of RStudio

If you’re using a local version of RStudio on your laptop (i.e. not accessed via the Analytical Platform control panel), you can import data from .csv files into Rstudio by clicking on the Environment tab and then the Import 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 can simply use the function read\_csv() from the readr package (which is included as part of tidyverse - see section 2.2):

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

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 cloud storage (Amazon s3)

Data that has been approved for storage on the Analytical Platform is generally stored in a data source (referred to as a ‘bucket’) on Amazon s3, which is the cloud storage solution used by the Analytical Platform. To import the data for this session from the ‘alpha-r-training’ s3 bucket use this command:

offenders <- botor::s3\_read("s3://alpha-r-training/intro-r-training/Offenders\_Chicago\_Police\_Dept\_Main.csv", read.csv)

Both this and the code shown above are suitable for csv files, so it is assumed by default that the first line of the file contains a header (header = T) and the columns are separated by a comma symbol (sep = “,”). The latter approach assumes that you have already loaded the readr package, as it is part of tidyverse - see section 2.2.

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 11 variables. To view this dataset, click the icon to the right of this information (or anywhere on that row), 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)

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 11 variables. Variables in a data frame are like columns in a table, and are stored as vectors. Also provided is some information about each variable (or vector) 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)

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 variable for the 500th observation:

offenders[500,4]

If you want the fourth variable for the 500th and 502nd observations you can use the concatenate (c) command:

offenders[c(500, 502),4]

If you want the first five variables for the 500th observation:

offenders[500,1:5]

The colon operator allows you to create sequences - in this case from 1 to 5, so here you will retrieve from the 1st to the 5th variables.

Dataframes in R are a collection of vectors where each vector is a column and represents a variable. To view a specific variable, for instance gender, you can also use a dollar sign as follows:

offenders$GENDER

The format is dataframe name, $, variable name. Note that a vector is returned.

### 2.5 Data classes

All variables have an associated class. The class will determine what calculations are possible 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” (see example checking the class of the WEIGHT variable below):

class(offenders$WEIGHT)

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 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")

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)

### 2.6 Some 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

### 2.7 Exercises

1. Find the mean and median of the AGE variable in the offenders dataset.
2. Find the max and min for the WEIGHT variable in the offenders dataset.
3. By changing the SENTENCE class to factor output the levels of this variable.

## 3 Data wrangling and ‘group by’ calculations

### 3.1 Select

We can keep only those variables we want from the offenders dataset using the select command from the dplyr package.

?dplyr::select

The use of double colons enables you to specify the package you are referring to before calling the function, hence avoiding using the wrong function if two functions have the same name and are from different packages. If the package isn’t specified by using the double-colon notation then R will use the function from your most recently loaded package and will warn you when you load a package if there is some overlap.

So, if we want to create a new dataset called offenders\_anonymous which only includes the variables representing date of birth, weight and number of previous convictions from the dataset offenders:

offenders\_anonymous <- dplyr::select(offenders, BIRTH\_DATE, WEIGHT, PREV\_CONVICTIONS)

The first argument within the select command specifies use of the offenders dataset. Following this we list the variables we want to keep.

A more popular way to obtain the same result is to use the pipe (%>%) operator:

offenders\_anonymous <- offenders %>%   
 dplyr::select(BIRTH\_DATE, WEIGHT, PREV\_CONVICTIONS)

Here the offenders data are ‘piped’ like water into the select command using the pipe symbol %>%. This is interpreted by R as the first argument of the select command so the offenders dataset is not specified within the select command. The pipe operator makes code more readable by allowing us to chain together multiple functions and means you don’t have to either create a new object each time you run a command or use nested functions (functions that are within other functions).

Let’s say that now we want the offenders\_anonymous dataset to be the same as the dataset offenders but without the names and addresses:

offenders\_anonymous <- offenders %>%   
 dplyr::select(-LAST, -FIRST, -BLOCK)

As we don’t want them, the variables listed within the select command now have minus signs before each of them.

### 3.2 Grouping and summarising data

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

* 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 produce summary statistics in a similar way to pivot tables in Excel. We can use the pipe (%>%) operator to chain these functions together so that we don’t have to create a new object each time we run each of the commands, and in a manner which makes the code easy to read.

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

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

Here R takes the offenders dataset, then (the pipe operator can be read as “then”) groups it first by REGION and then by GENDER and then outputs the mean number of previous convictions by REGION and GENDER. The mean number of previous convictions variable created we’ve decided to call “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 to add a new variable that we decide to call “Count”” that provides the counts by REGION and GENDER we can rerun as follows using the pipe operator:

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

It is important to pay attention to the way in which the data have been grouped. The regional\_gender\_average dataset is currently grouped by REGION and GENDER. If we run it through summarise as is, then the result will be grouped by the first grouping variable, which in this case is REGION:

regional\_gender\_average %>%   
 dplyr::summarise(Count = n())

But if we want to count all the rows in the regional\_gender\_average dataset with the grouping removed we add in the ungroup() function:

regional\_gender\_average %>%   
 dplyr::ungroup() %>%   
 dplyr::summarise(Count = n())

### 3.3 Filter

If you would like to produce statistics for a subset of rows or observations, a good function to use 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 %>%   
 dplyr::group\_by(SENTENCE) %>%   
 dplyr::summarise(Count = n())

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 %>%   
 dplyr::filter(SENTENCE == "Court\_order" & AGE > 50) %>%   
 dplyr::group\_by(REGION, GENDER) %>%   
 dplyr::summarise(Ave = mean(PREV\_CONVICTIONS))

### 3.4 Rename

We can rename variables using the dplyr function rename(). Let’s amend our section 3.1 coding in creating the offenders\_anonymous dataset so that BIRTH\_DATE is instead called “DoB”.

offenders\_anonymous <- offenders %>%  
 dplyr::select(BIRTH\_DATE, WEIGHT, PREV\_CONVICTIONS) %>%  
 dplyr::rename(DoB = BIRTH\_DATE)

Within the rename function, the new name “DoB” is specified on the left and the old name on the right of the equal sign. If you also wanted to rename PREV\_CONVICTIONS to “Num\_prev\_convictions” to make it easier to understand then simply add this as an extra argument within the rename function, with the two arguments separated by a comma:

offenders\_anonymous <- offenders %>%  
 dplyr::select(BIRTH\_DATE, WEIGHT, PREV\_CONVICTIONS) %>%  
 dplyr::rename(DoB = BIRTH\_DATE, Num\_prev\_convictions = PREV\_CONVICTIONS)

### 3.5 Mutate

You can create new variables and perform calculations on variables using the dplyr command mutate().

?mutate

So if we wanted to amend our coding to include a new derived variable weight\_kg in the offenders\_anonymous dataset:

offenders\_anonymous <- offenders %>%  
 dplyr::select(BIRTH\_DATE, WEIGHT, PREV\_CONVICTIONS) %>%  
 dplyr::rename(DoB = BIRTH\_DATE, Num\_prev\_convictions = PREV\_CONVICTIONS) %>%  
 dplyr::mutate(weight\_kg = WEIGHT \* 0.454)

You can download the Data Transformation Cheat Sheet (and other cheatsheets) at: <https://www.rstudio.com/resources/cheatsheets/>

### 3.6 If\_else

Another useful function found in the dplyr package is if\_else, which works in a similar way to if statements in Excel. This uses a logical statement to determine the output. The below code uses the if\_else function to identify offenders who have weight under 170lbs, the mutate function being used together with it to add a variable in to the ‘offenders’ dataset which is 1 if they are under 170lbs and 0 if they are over 170lbs.

offenders <- offenders %>%   
 dplyr::mutate(weight\_under\_170 = if\_else(WEIGHT<170,1,0))

### 3.7 Exercises

1. Using group\_by and summarise, calculate the average and median age for females in the West.
2. Using select and filter produce a table of offender’s genders who are over 2m tall (note that the heights are currently in cm).
3. Produce a table showing the counts of height (including missing values).

Extension: 4. Create a new dataset containing PREV\_CONVICTIONS and SENTENCE variables, rename SENTENCE as sentence\_type, and create a new variable num\_convictions that is PREV\_CONVICTIONS + 1 (to take account of the latest conviction).

## 4 Dates

### 4.1 Manipulating dates

As you might have noticed, BIRTH\_DATE in the offenders dataset currently has class character. To be able to manipulate dates in date format, we first need to convert the data to have class date.

In this section, we are going to use a package from tidyverse called lubridate to enable R to recognize and manipulate dates. First, we need to load the package:

library(lubridate)

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:

lubridate::today()

If you have a read of the help file, you’ll see lubridate has a number of functions such as dmy(), myd() etc whose name models the order in which the year (‘y’), month (‘m’) and day (‘d’) elements appear in the character string to be parsed.

We can therefore make a new date variable (called DoB\_formatted) with class date as follows, and then check the class of the new column:

offenders<- offenders %>%   
 dplyr::mutate(DoB\_formatted = lubridate::mdy(BIRTH\_DATE))  
  
class(offenders$DoB\_formatted)

The function mdy() specifies the format that the date in column BIRTH\_DATE is currently in so R knows where to find the day, month and year needed to create a date.

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

offenders <- offenders %>%   
 dplyr::mutate(day = lubridate::day(DoB\_formatted))  
  
offenders <- offenders %>%  
 dplyr::mutate(quarter = lubridate::quarter(DoB\_formatted))  
  
offenders <- offenders %>%  
 dplyr::mutate(year = lubridate::year(DoB\_formatted))  
  
offenders <- offenders %>%  
 dplyr::mutate(month = lubridate::month(DoB\_formatted))  
  
offenders <- offenders %>%  
 dplyr::mutate(weekday = weekdays(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 <- offenders %>%   
 dplyr::mutate(days\_before\_2000 = lubridate::ymd("2000-01-01") - DoB\_formatted)

### 4.2 Exercises

1. Read in dataset ‘FTSE\_12\_14.csv’ and convert the variable date to class date. (To read in the data, use the code given in the participant code script)
2. Add a variable called day with the day of the week, and another variable called daily\_performance for how much the share price has increased or decreased that day (close price - open price).
3. Work out which day of the week has the highest mean performance.

## 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 information on whether the offenders faced trial. Use either of the following commands, depending on whether you’re using the Analytical Platform version of RStudio or a local version:

offenders\_trial <- botor::s3\_read("s3://alpha-r-training/intro-r-training/Offenders\_Chicago\_Police\_Dept\_Trial.csv", read.csv)

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

We merge the datasets with offenders using the combination of fields that together form a unique identifier. But first we need to rename DoB to BIRTH\_DATE in the offenders\_trial dataset:

offenders\_trial <- offenders\_trial %>% dplyr::rename(BIRTH\_DATE=DoB)

Now the variables that together form a unique identifier have the same names, we can do the merge:

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

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 <- offenders %>%   
 dplyr::filter(GENDER == "MALE")  
women <- offenders %>%   
 dplyr::filter(GENDER == "FEMALE")  
  
rejoined <- dplyr::bind\_rows(men, women)

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

nrow(rejoined)   
  
nrow(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). The missing data functions we’ll use in this section recognize both these types.

We can look at the HEIGHT variable as previously:

height\_table <- offenders %>%   
 dplyr::group\_by(HEIGHT) %>%   
 dplyr::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 create a logical vector showing whether each HEIGHT observation is missing (TRUE) or not (FALSE):

is.na(offenders$HEIGHT)

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 <- offenders %>%   
 dplyr::filter(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 file in your working directory.

### 5.4 Exercises

1. Create 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.

Extension: 3. Using offenders create a new variable HEIGHT\_NEW which is as HEIGHT except with the missing values replaced by the average height (hint: you can make use of the replace function).

## 6. Extra Resources

There are lots of resources that can help you develop your R knowledge, but below are a few that are particularly helpful:

* RStudio has developed a list of cheatsheets which give quick overviews of the functions contained in different packages and can be quickly referred to - see: <https://www.rstudio.com/resources/cheatsheets/> Some can be accessed directly through the top menu help > Cheatsheets e.g. ‘Data Transformation with dplyr’.
* RStudio also lists some useful free resources at: <education.rstudio.com>
* The ‘R for Data Science’ online book: <r4ds.had.co.nz/>, written by Hadley Wickham, a data scientist at RStudio, who developed the tidyverse package. It gives a really good overview of R and how his package works with it.

For further lists of useful resources please see: <https://trello.com/b/D5pSkqnT/online-analytical-training>