Intro R training

## Basic setup

This document provides accompanying training material used in the Introduction to R Training session, conducted by the ASD R training group. 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>. Also make sure step 2.3.1 has been completed otherwise you will have difficulty importing files at later stages of this course.

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, please 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>

## 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 in ASD 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) 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.

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 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.

### 1.7 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

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, 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, 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 is already 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 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 ✔ purrr 0.2.4  
FALSE ✔ tibble 1.3.4 ✔ dplyr 0.7.4  
FALSE ✔ tidyr 0.7.2 ✔ stringr 1.2.0  
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()

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

It is important to be able to import data both from the analytical platform amazon server and other servers we work with.

#### Importing data not on the analytical platform amazon server

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")

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 either one of these commands:

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

## using csv (or similar) method, reading directly to R supported

offenders<-s3tools::read\_using(FUN=read.csv, s3\_path = "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 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)

## 'data.frame': 1413 obs. of 11 variables:  
## $ LAST : Factor w/ 927 levels " CORY","ABADIA",..: 723 530 327 723 151 838 67 556 639 399 ...  
## $ FIRST : Factor w/ 567 levels " HARDING","AARIN",..: 307 400 180 304 436 34 71 393 120 224 ...  
## $ BLOCK : Factor w/ 1167 levels "0000X E 100TH ST",..: 141 160 163 175 183 223 242 259 275 285 ...  
## $ GENDER : Factor w/ 2 levels "FEMALE","MALE": 2 2 2 2 2 2 2 2 2 2 ...  
## $ REGION : Factor w/ 4 levels "East","North",..: 4 1 3 2 3 2 1 1 3 1 ...  
## $ BIRTH\_DATE : Factor w/ 1347 levels "01/01/1952","01/01/1966",..: 623 134 817 147 387 1191 1136 1234 138 626 ...  
## $ 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 : Factor w/ 3 levels "Court\_order",..: 1 2 1 1 2 2 2 1 1 2 ...  
## $ AGE : int 58 59 43 54 34 50 32 43 34 18 ...

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. 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)

## LAST FIRST BLOCK GENDER   
## WILLIAMS: 25 MICHAEL: 43 053XX S INDIANA AVE : 12 FEMALE: 32   
## JONES : 23 ROBERT : 33 030XX S ARCHER AVE : 10 MALE :1381   
## BROWN : 18 JAMES : 27 004XX S CLARK ST : 7   
## JOHNSON : 16 ANTHONY: 25 008XX W 51ST PL : 7   
## SMITH : 15 CHARLES: 21 027XX W HARRISON ST : 7   
## HARRIS : 14 DAVID : 20 035XX W ROOSEVELT RD: 7   
## (Other) :1302 (Other):1244 (Other) :1363   
## REGION BIRTH\_DATE HEIGHT WEIGHT   
## East :352 06/14/1969: 3 Min. : 24.0 Min. :100.0   
## North:358 09/04/1984: 3 1st Qu.:200.0 1st Qu.:160.0   
## South:378 09/04/1988: 3 Median :200.0 Median :180.0   
## West :325 09/12/1962: 3 Mean :209.1 Mean :188.2   
## 01/01/1971: 2 3rd Qu.:236.0 3rd Qu.:210.0   
## 01/04/1956: 2 Max. :239.0 Max. :601.0   
## (Other) :1397 NA's :24   
## PREV\_CONVICTIONS SENTENCE AGE   
## Min. : 0.00 Court\_order:856 Min. :15.00   
## 1st Qu.: 1.40 Prison\_<12m:417 1st Qu.:34.00   
## Median : 9.80 Prison\_12m+:140 Median :43.00   
## Mean :12.83 Mean :43.88   
## 3rd Qu.:22.40 3rd Qu.:52.00   
## Max. :39.20 Max. :94.00   
##

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]

## [1] MALE  
## Levels: FEMALE MALE

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

offenders[c(500, 502),4]

## [1] MALE MALE  
## Levels: FEMALE MALE

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

offenders[500,1:5]

## LAST FIRST BLOCK GENDER REGION  
## 500 VAINISI ROBERT 059XX W 63RD PL MALE West

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 columns.

Dataframes/tibbles 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 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 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” (see example checking the class of the WEIGHT variable below):

class(offenders$WEIGHT)

## [1] "integer"

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)

### 2.6 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$wt\_under\_170 <- ifelse(offenders$WEIGHT<170, 1, 0)

### 2.7 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

### 2.8 Exercises

1. Find the mean, median, max and min for variables AGE and WEIGHT in the dataset offenders.
2. By changing the SENTENCE class to factor output the levels of this variable.
3. Create a new variable called ‘height\_under\_150’ which is 1 if under 150 cm and 0 otherwise.

## 3 Data wrangling and ‘group by’ calculations

### 3.1 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

Throughout this course you may have noticed double colons being used following the name of a package as above. This specifies the package you are refering to before calling the function, hence avoiding using the wrong function if two from different packages you have loaded have the same name. In general 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. We will not specify packages throughout this code due to the nature of the course, but it is good practice to specify packages in this way if you are sharing code or moving between projects.

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 these functions together using the pipe (%>%) operator which makes code more readable and means you don’t have to create a new object each time you run a command. Using the pipe operator reduces the amount of nested functions (functions that are within other functions) and allows us to chain together dplyr data wrangling functions. The pipe operator can be read as “then” and allows us to go from one step to another easily in dplyr so we can, for example:

* filter our data frame to only focus on a few rows then
* group\_by another variable to create groups then
* summarize this grouped data to calculate the mean for each level of the group.

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))

The pipe operator simply passes through the object on the left hand side as the first argument of the following function. If we did not use the pipe operator to obtain the mean number of previous convictions with breakdown by REGION and GENDER the R code would look like this:

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

Using nested functions like this looks more complicated and is harder to follow than if we use the pipe operator, this becomes more clear as extra calculations are added.

In the above code R is taking the ‘offenders’ dataset, grouping it first by REGION and then by GENDER (note that the data is “piped” like water into the dplyr::group\_by command using the pipe symbol ‘%>%’ instead of specifying the ‘offenders’ dataset as the first argument within the dplyr::group\_by command), and then outputting the mean number of previous convictions by REGION and GENDER. The new mean number of previous convictions variable 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 called ‘Count’ that provides the counts by REGION and GENDER we can rerun as follows using the pipe operator:

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 run it through dplyr::sumuraise again the result will be grouped by the first grouping variable set which is REGION For example, the summarise() function applied below counts the rows in each of the first group by category.

regional\_gender\_average %>%   
 summarise(n = n())

But if we wanted to count all the rows in the dataset with the grouping removed we add in the dplyr::ungroup() function as below:

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

## # A tibble: 1 x 1  
## n  
## <int>  
## 1 8

### 3.2 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 %>%   
 group\_by(SENTENCE) %>%   
 summarise(Count = n())

## # A tibble: 3 x 2  
## SENTENCE Count  
## <fctr> <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))

### 3.3 Select

We can also use the select() command from the dplyr package to choose just the variables from the ‘offenders’ dataset that we want. So if we want to create a new dataset without the names and addresses:

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

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

Let’s say that now we want to restrict this dataset to just include BIRTH\_DATE, WEIGHT and number of previous convictions.

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

### 3.4 Rename

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

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

If you also wanted to rename PREV\_CONVICTIONS to Num\_prev\_convictions to make it easier to understand then:

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

1. Using group\_by and summarise, calculate the average and median age for females in the West.
2. How many have heights of less than 2 metres, what are their (recorded) heights and gender(s)?
3. Produce a table showing the counts of height (including missing values).
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 a class of character (or factor depending how you read the data in). To be able to manipulate this data as a date in R, we first need to convert it to class date.

In all of this section, we are going to use a package from tidyverse designed called lubridate, there are also ways to do the same thing using base R. First, we need to load the package:

library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## 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:

today()

## [1] "2019-03-28"

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 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<- mutate(offenders, DoB\_formatted = 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.

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 <- mutate(offenders, day = day(DoB\_formatted))  
  
offenders <- mutate(offenders, quarter = quarter(DoB\_formatted))  
  
offenders <- mutate(offenders, year = year(DoB\_formatted))  
  
offenders <- mutate(offenders, month = month(DoB\_formatted))  
  
offenders <- mutate(offenders, 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 <- mutate(offenders, days\_before\_2000 = 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 using s3, use s3tools::s3\_path\_to\_full\_df(“alpha-everyone/R\_training\_intro/FTSE\_12\_14.csv”) or s3tools::read\_using(FUN=read.csv, s3\_path=“alpha-everyone/R\_training\_intro/FTSE\_12\_14.csv”)).
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.

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

Or using one of the analytical platform options:

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

## using csv (or similar) method, reading directly to R supported

offenders\_trial <-s3tools::read\_using(FUN=read.csv, s3\_path = "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. 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 <- 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 %>% filter(GENDER == "MALE")  
women <- offenders %>% filter(GENDER == "FEMALE")  
rejoined <- 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).

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 <- offenders %>% 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 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.

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 will need to use the ifelse and is.na() functions).

## 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:

* DataCamp is a website which hosts multiple online courses that teach coding. Their ‘Introduction to R’ course is free to complete and provides a broader overview in the basic concepts for coding in R. A link to the course can be found here: <https://www.datacamp.com/courses/free-introduction-to-r>.
* Another good resourse is the ‘R for Data Science’ online book: <r4ds.had.co.nz/>, written by Hadley Wickham, who is a data scientist at RStudio, who developed the tidyverse package that we introduced earlier. It gives a really good overview of R and how his package works with it.
* RStudio has also developed a list of ‘cheatsheets’ which give quick overviews of the functions contained in different packages, which can be quickly referred to: <https://www.rstudio.com/resources/cheatsheets/> Some can be accessed directly through the top menu help > Cheatsheets e.g. ‘Data Transformation with dplyr’.

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