R Beginners Exercise 2: Data Processing

# Introduction

Welcome to R for Beginners Exercise 2! This notebook contains the exercises for the lesson that we will be looking at during exercise breaks throughout the course as well as being a work space for you to use during the session!

To execute a line of code, click on it and press *Ctrl + Enter*.

To execute a chunk of code, click the green run button at the top right corner of the code chunk or highlight the entire code chunk and press *Ctrl + Enter*.

# 2.1 Import dataset

R supports importing datasets from a range of file types. Some common file types supported by R are .csv, .txt, .xlxs. There are two methods to import a dataset.

* Through the “Import Dataset” button in the Environment window (top right).
* Through command *(more information regarding this method can be found in the* [*R documentation*](https://cran.r-project.org/doc/manuals/R-data.html#XML)*)*

*\* To import data through command, you will have to first set working directory to the folder where your data files are stored. However, since this notebook is cloned from GitLab and is opened in a project space, you can skip this step.*

## 2.1.1 Import from CSV file

students = read.csv("Ex2-Students\_CSV.csv")  
students

## ID First.Name Last.Name  
## 1 D001 <NA> Smith  
## 2 D002 Anne Clare  
## 3 D003 Charlotte <NA>  
## 4 D004 Thomas Train

## 2.1.2 Import from TXT file

menu = read.delim("Ex2-Menu\_TXT.txt") # with tab character as the delimiter  
menu

## Item Price  
## 1 Burger 10.90  
## 2 Fries 4.45  
## 3 Juice NA  
## 4 Salad 9.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

## 2.1.3 Import from Excel file

library(readxl) # the package which enables Excel file importation  
students2 = read\_xlsx("Ex2-Students\_Excel.xlsx")  
students2

## # A tibble: 4 × 3  
## ID `First Name` `Last Name`  
## <chr> <chr> <chr>   
## 1 D001 John Smith   
## 2 D002 Anne Clare   
## 3 D003 Charlotte Brey   
## 4 D004 Thomas Train

# 2.2 Dataset overview

## 2.2.1 View dataset in new window

Use the view() function to view the students dataset in another window.

# The package which contains the view() function to display dataset in a separate window  
library(tibble)  
  
# Write your code below  
view(students)

## 2.2.2 View columns of dataset in console

Use the glimpse() function to print all the columns of the students dataset to console.

# Write your code below  
glimpse(students)

## Rows: 4  
## Columns: 3  
## $ ID <chr> "D001", "D002", "D003", "D004"  
## $ First.Name <chr> NA, "Anne", "Charlotte", "Thomas"  
## $ Last.Name <chr> "Smith", "Clare", NA, "Train"

## 2.2.3 View column names in console

Use the names() function to print all the column names of the students dataset to console.

# Write your code below  
names(students)

## [1] "ID" "First.Name" "Last.Name"

\* In R, you can view column names of a dataset when writing code by typing the dollar sign, $, after the dataset’s name.

## 2.2.4 Find datatype of a column

The class() function in R can be used to find the datatype of a specific column in a table, e.g. class(tableName$columnName). Use the class() function to find the datatype of the Price column in the menu dataset.

# Write your code below  
class(menu$Price)

## [1] "numeric"

To find the datatype of all columns in a table, R supports the sapply(list, function) function which applies the same function to a list of items. The code below shows an example of using this function to find the datatype of all columns in the menu table.

sapply(menu, class)

## Item Price   
## "character" "numeric"

Try to find the datatype of all columns in the students table by using the sapply() function.

# Write your code below  
sapply(students, class)

## ID First.Name Last.Name   
## "character" "character" "character"

## 2.2.5 Summary of dataset

Use the summary() function to get a summary of all columns in the menu dataset.

# Write your code below  
summary(menu)

## Item Price   
## Length:6 Min. : 3.330   
## Class :character 1st Qu.: 4.450   
## Mode :character Median : 9.500   
## Mean : 7.816   
## 3rd Qu.:10.900   
## Max. :10.900   
## NA's :1

## 2.2.6 Find unique values in a column

The unique() function in R can be used to find all unique values in a specific column in a table, e.g., unique(tableName$columnName). Use the unique() function to find all values in the Price column in the menu dataset, without duplication.

# Write your code below  
unique(menu$Price)

## [1] 10.90 4.45 NA 9.50 3.33

By using the sapply() function, find the unique values in each column in the menu table.

# Write your code below  
sapply(menu, unique)

## $Item  
## [1] "Burger" "Fries" "Juice" "Salad" "Chicken" "Drinks"   
##   
## $Price  
## [1] 10.90 4.45 NA 9.50 3.33

## 2.2.7 Incomplete cases in dataset

In R, incomplete cases are rows in dataset that have Na value(s). These incomplete cases can be viewed using the function complete.cases.

For example, we can view the incomplete cases in the menu dataset by executing the code below.

menu[!complete.cases(menu),]

## Item Price  
## 3 Juice NA

Breaking down the code written, a condition is being used with the [] operator. The condition states ‘rows in menu that are not a complete case’, utilising the NOT logical operator and the complete.cases function. Note that the , sign within the [] operator is mandatory since menu is a 2-dimensional dataframe. The second index has been left out on purpose to include all columns in the result.

With reference to the syntax shown above, find the incomplete cases in students

# Write your code below  
students[!complete.cases(students),]

## ID First.Name Last.Name  
## 1 D001 <NA> Smith  
## 3 D003 Charlotte <NA>

# 2.3 Data ordering

The order of vector elements can be easily reordered using the function sort() in R. It can be applied to all basic datatypes to order data in numerical order (numeric, integer, complex, logical datatypes) or alphabetical order (character datatypes). It has a parameter, decreasing, to specify whether to sort a vector in ascending or descending order, e.g., sort(someVector, decreasing = TRUE).

\* Executing the code sort(someVector) will only print the sorting result to console. To save the sorted vector, you would need to write someVector = sort(someVector).

Execute the numbers vector defined below, then sort it in ascending order.

numbers = c(4, 3, 5, 1, 2)  
  
# Write your code below  
numbers = sort(numbers)

Execute the words vector defined below, then sort it in descending alphabetical order.

words = c("table", "water", "chair", "ruler")  
  
# Write your code below  
words = sort(words, decreasing = TRUE)

# 2.4 Dealing with incomplete cases

Missing values in R are very problematic as R cannot process a function correctly with their presence, and may even run into error. In order to obtain accurate results, these incomplete cases will need to be dealt with.

## 2.4.1 Drop all incomplete cases

The simplest way to deal with missing value(s) is to drop (remove) all the incomplete cases. Extending from the syntax to find incomplete cases in [2.2.7 Incomplete cases in dataset](#incomplete-cases-in-dataset), we can drop all incomplete cases by omitting the use of NOT operator such as:

menu[complete.cases(menu),]

## Item Price  
## 1 Burger 10.90  
## 2 Fries 4.45  
## 4 Salad 9.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

Try using this method to output a table which drops all the incomplete cases in students.

# Write your code below  
students[complete.cases(students),]

## ID First.Name Last.Name  
## 2 D002 Anne Clare  
## 4 D004 Thomas Train

A simpler method uses the function na.omit(). Using this function, try to obtain the same output table as above.

# Write your code below  
na.omit(students)

## ID First.Name Last.Name  
## 2 D002 Anne Clare  
## 4 D004 Thomas Train

As you might notice, dealing with missing value(s) by dropping all the incomplete cases ignores a lot of rows in the dataset. Therefore, this method is not recommended because it can greatly affect the reliability of the analysis.

## 2.4.2 Drop incomplete cases in specific column

A better way to deal with missing value(s) is to remove only rows with a missing value in the variable we are interested in. This minimises the number of rows being ignored. The method to do so is similar to the first method in the previous section, with the addition of the name of the column that you want to drop missing values from in the condition.

menu[complete.cases(menu$Price),]

## Item Price  
## 1 Burger 10.90  
## 2 Fries 4.45  
## 4 Salad 9.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

Apply this method on the students dataset to output a table where rows with missing value in the First.Name column are removed.

# Write your code below  
students[complete.cases(students$First.Name),]

## ID First.Name Last.Name  
## 2 D002 Anne Clare  
## 3 D003 Charlotte <NA>  
## 4 D004 Thomas Train

## 2.4.3 Replace the missing value(s)

If you know your data well enough, you can replace the missing value(s) in the dataset. This can be done manually by assigning a new value to the cell. Taking the dataset menu as an example, the cell with missing value has index [3, 2], a new value can be assigned to it as follows:

menu[3, 2] = 4.50  
menu # Check to make sure the missing value is being replaced correctly.

## Item Price  
## 1 Burger 10.90  
## 2 Fries 4.45  
## 3 Juice 4.50  
## 4 Salad 9.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

For a small dataset, the method above is plausible. For a larger dataset, the better way is to replace missing value(s) in a batch using the function replace\_na in the dbplyr package.

library(tidyr) # The package which includes the replace\_na function.   
replace\_na(menu, list(Price = 4.50))

## Item Price  
## 1 Burger 10.90  
## 2 Fries 4.45  
## 3 Juice 4.50  
## 4 Salad 9.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

This method should be used with caution as it replaces all NA values in the column with the value specified. Run the code chunk below which shows an example of such an issue.

# Creating the dataset for illustration purposes.   
menu\_example = menu  
menu\_example[1:4, 2] = NA  
menu\_example

## Item Price  
## 1 Burger NA  
## 2 Fries NA  
## 3 Juice NA  
## 4 Salad NA  
## 5 Chicken 10.90  
## 6 Drinks 3.33

# Applying the function  
replace\_na(menu\_example, list(Price = 4.50))

## Item Price  
## 1 Burger 4.50  
## 2 Fries 4.50  
## 3 Juice 4.50  
## 4 Salad 4.50  
## 5 Chicken 10.90  
## 6 Drinks 3.33

The students2 dataset is a complete version of the students dataset, refer to it to apply the following changes to the students dataset.

1. Replace the missing value in the First.Name column using the first method.
2. Replace the missing value in the Last.Name column using the second method.

# Replacing with first method  
students[1, 2] = "John"  
  
# Replacing with second method  
replace\_na(students, list(Last.Name = "Brey"))

## ID First.Name Last.Name  
## 1 D001 John Smith  
## 2 D002 Anne Clare  
## 3 D003 Charlotte Brey  
## 4 D004 Thomas Train

## 2.4.4 Ignore missing value(s) in function

When applying aggregate functions to a dataset with missing value(s), the value NA will be returned. In order to obtain a useful output, we can ask these functions to ignore the missing value(s) by setting the na.rm parameter to TRUE.

\* Aggregate functions: sum, mean, median, min, max, variance, range, etc.

numbers = c(2, 3, 5, 8, NA, 4, 7)  
  
# Using the default value assigned to the na.rm parameter  
sum(numbers)

## [1] NA

# Setting the na.rm parameter to TRUE to ignore missing value(s)  
sum(numbers, na.rm = TRUE)

## [1] 29

Calculate the mean prices in the menu\_example dataset.

\* Use datasetName$colname to specify which column to apply the function on.

# Write your code below  
mean(menu\_example$Price, na.rm = TRUE)

## [1] 7.115