Data Exploration 1 : Heart Failure Example

Leonard Wee

24-1-2022

Outline of this document

- How simple functions make you code much cleaner and more reproducible / re-usable
- Learning how to use the source function in R to re-use functions from other R scripts
- Importing data and clearly showing basic adjustments you made to the data in R
- Example of a readable and easily explainable block of R code to import and recode some data

Re-usable functions

One of the most crucial tips you can start to practice is to break down processing steps into simple parts. The temptation is to make one gigantic monolithic script that tries to do everything on one click, and the result is usually impossible to debug or update.

Specifically, try to **put repetitive simple steps in re-usable functions**, and then this function can be re-called many times from many scripts.

Here is an example of trying to install and load packages in your R code.

```
# ---- eval = FALSE means this code block will actually not execute when knitting the markdown

#lets try to install and load dplyr, reshape, reshape2 and magrittr

install.packages("dplyr")
library("dplyr")

install.packages("reshape")
library("reshape")

install.packages("reshape2")
library("reshape2")

install.packages("magrittr")
library("magrittr")

#notice that we are having to copy paste the same operation several times to load our favourite package
```

Consider the this step below as an alternative, the end result is the same but which one is easily transferable to new code and new situations? Which is easier to read and inspect which packages are actually needed?

```
source("handy_dandy_functions.R") #the **source** function executes the contents of a separate R file

list.of.packages <- c("dplyr", "reshape","reshape2", "magrittr", "httr") #I define a list of packages I

installRequiredPackages(list.of.packages) #this calls the function that installs them one by one
```

```
## [1] "dplyr"    "reshape"  "reshape2" "magrittr" "httr"
```

Structure of R functions

All user functions in R have exactly the same structure :

```
add_two_numbers_together <- function(number1, number2) {
  #do some kind of action using the arguments
  output_variable = number1 + number2
  #save it as a new variable internally in the function
  return(output_variable)
}

add_two_numbers_together(2,3)
```

```
## [1] 5
```

```
#what do you think the output will be? Try running this code block in the console
```

Here is an actually useful function such as converting the values in your column from degrees F to degrees C:

```
fahrenheit_to_celsius <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(temp_C)
}

my_temperatures <- c(32, 55, 98.6, 104) #pretend this is your data

fahrenheit_to_celsius(32) #doing it one by one
```

```
## [1] 0
```

```
sapply(my_temperatures, fahrenheit_to_celsius) #compare doing this in the smart R way
```

```
## [1]  0.00000 12.77778 37.00000 40.00000
```

Think of situations in your data preparation where defining a re-usable function and then applying it consistently will the quality of your code For example, data transformations such as taking the logarithm or cube root of values, or making sure that all date formats are consistent in your data?

The source command

A **good optimization step** for readibility of your code is to put useful functions into another document. This means functions of a related type are clustered together and easily editable. It means you do not have to scan hundreds of lines of code to look for something. The R command to learn here is ***source**, and the argument it needs is the R script file (or the system path including the R script file) that contains your re-usable functions.

```
source("handy_dandy_functions.R") #this was used in the above example for installing packages
```

Summarizing the data

Stable ways of importing data

The most traditional way to pull data into R is from a local file in your system. Here is an example taking in a CSV. Again, we try to write this in a way that will be easy to read and re-use. Here is an example where the path to where the file is stored and the file itself is joined using the **file.path** command.

```
pathToData <- 'C:/Users/leonard.wee/OneDrive - Maastro - Clinic/Documents/GitHub/EPI4932_Clinical_Data_S
#this is merely an example of where my data may be

pathToData <- '.' #this means "the same folder as where my markdown script is currently sitting"
rawHeartFailure <- read.csv( file.path(pathToData,'Heart failure.csv') )

#to learn more about the **read.csv** function, try typing "?read.csv" into the console
```

There are other ways to serve up data for research, and increasingly we are pulling prepared data directly from webservers. **This allows us to split the job of data curation from the data analysis**.

Compare this to a paradigm where everything from raw collection, to cleaning to final analysis has to be done by the same person? What is your opinion - do you think it is a good idea to split out the data science lifecycle so different jobs might be done by different people?

These web data servers are internally version controlled and auditable, and presents a tightly controlled interface to serve up data on-demand to analysts. This allows it to be more self-service for human users, and it enable software applications to get data by themselves.

Here is an example that uses something called an application programming interface (API, but more precisely it is a RESTful API) provided by the web data server. APIs are designed so that users can send micro-programs within a tightly prescribed way to search for and filter data directly from the server.

This is an example of an (out od date) API data request directed to a specific National Health Service data server in the UK. But it still works for play/test purposes. The documentation is at https://coronavirus.data.gov.uk/metrics/area_type/region.

```
#define data endpoint which will be a RESTful API on the web
endpoint <- 'https://api.coronavirus.data.gov.uk/v1/data'

#customize filters for the query:
AREA_TYPE = "utla" #upper tier local authority (UTLA)
AREA_NAME = "oxfordshire" #is an instance of an UTLA

filters <- c(
  sprintf("areaType=%s", AREA_TYPE),
```

```
  sprintf("areaName=%s", AREA_NAME)
)

#customize the structure of the data return:
structure <- list(
  date = "date",
  name = "areaName",
  areaCode = "areaCode",
  dailyCases = "newCasesByPublishDate",
  cumDeaths28DaysSincePos = "cumDeaths28DaysByPublishDate",
  newDeaths28DaysSincePos = "newDeaths28DaysByPublishDate"
)

#using best practices I made a reusable function to get data consistent from this API - see the file I
#note that you do not necessarily need to know details of the API scripting - you can re-use my functio
#care of most things for you
coronavirusData <- getDataUsingOpenApiMethod(endpoint, filters, structure)
```

```
## [1] "https://api.coronavirus.data.gov.uk/v1/data?filters=areaType%3Dutla%3BareaName%3Doxfordshire&st
```

```
#as an exercise, see if you can modify one of the lines above to request for the Covid-19 data from a d
```

```
##         date        name  areaCode dailyCases cumDeaths28DaysSincePos
## 1 2023-02-02 Oxfordshire E10000025        276                    1371
## 2 2023-02-01 Oxfordshire E10000025          0                    1359
## 3 2023-01-31 Oxfordshire E10000025          0                    1359
## 4 2023-01-30 Oxfordshire E10000025          0                    1359
## 5 2023-01-29 Oxfordshire E10000025          0                    1359
## 6 2023-01-28 Oxfordshire E10000025          0                    1359
##   newDeaths28DaysSincePos
## 1                      12
## 2                       0
## 3                       0
## 4                       0
## 5                       0
## 6                       0
```

```
##            date        name  areaCode dailyCases cumDeaths28DaysSincePos
## 1013 2020-04-26 Oxfordshire E10000025         44                      NA
## 1014 2020-04-25 Oxfordshire E10000025         33                      NA
## 1015 2020-04-24 Oxfordshire E10000025         22                      NA
## 1016 2020-04-23 Oxfordshire E10000025         26                      NA
## 1017 2020-04-22 Oxfordshire E10000025         48                      NA
## 1018 2020-04-21 Oxfordshire E10000025         38                      NA
## 1019 2020-04-20 Oxfordshire E10000025         44                      NA
## 1020 2020-04-19 Oxfordshire E10000025         47                      NA
## 1021 2020-04-18 Oxfordshire E10000025         41                      NA
## 1022 2020-04-17 Oxfordshire E10000025         55                      NA
##      newDeaths28DaysSincePos
## 1013                      NA
## 1014                      NA
## 1015                      NA
```

```
## 1016                    NA
## 1017                    NA
## 1018                    NA
## 1019                    NA
## 1020                    NA
## 1021                    NA
## 1022                    NA
```

Preparing data for transparency and ease of use

We are fortunate with the heart failure data that we imported, because we have the coding dictionary immediately. See the word/pdf document in this repo. However, note what happens if you want to get a quick overview of the data. What do you notice about the output below?

```
summary(rawHeartFailure)
```

```
##       Age            Sex            ChestPainType        RestingBP
##  Min.   :28.00   Length:918         Length:918         Min.   :  0.0
##  1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0
##  Median :54.00   Mode  :character   Mode  :character   Median :130.0
##  Mean   :53.51                                         Mean   :132.4
##  3rd Qu.:60.00                                         3rd Qu.:140.0
##  Max.   :77.00                                         Max.   :200.0
##   Cholesterol      FastingBS       RestingECG            MaxHR
##  Min.   :  0.0   Min.   :0.0000   Length:918         Min.   : 60.0
##  1st Qu.:173.2   1st Qu.:0.0000   Class :character   1st Qu.:120.0
##  Median :223.0   Median :0.0000   Mode  :character   Median :138.0
##  Mean   :198.8   Mean   :0.2331                      Mean   :136.8
##  3rd Qu.:267.0   3rd Qu.:0.0000                      3rd Qu.:156.0
##  Max.   :603.0   Max.   :1.0000                      Max.   :202.0
##  ExerciseAngina        Oldpeak          ST_Slope          HeartDisease
##  Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000
##  Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000
##  Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000
##                     Mean   : 0.8874                      Mean   :0.5534
##                     3rd Qu.: 1.5000                      3rd Qu.:1.0000
##                     Max.   : 6.2000                      Max.   :1.0000
```

Let's make this work in a more data science kind of way. First, we notice from the column names that we don't know what **data type** we expect to see in each column - is it numerical continuous, is it numerical but only zero and one therefore binary, or what?

Here is a way to rename columns in a block, that makes your code highly readible and easy to follow what you have done, plus at the same time makes it clearer what kind of data to expect per column.

```
#robust and clear example of block renaming of columns when the dictionary is known
#explicit renaming in a block is much easier to read and maintain than one by one
recodedHeartFailure <- dplyr::rename(rawHeartFailure,
                                     num_age_in_years = Age,
                                     bin_sex_is_male = Sex,
                                     fac_chest_pain_type = ChestPainType,
                                     num_rest_blood_press_mmHg = RestingBP,
                                     num_serum_cholestrol_mm_per_dl = Cholesterol,
```

```
                                    bin_fasting_blood_gluc_gt_120mg_per_dl = FastingBS,
                                    fac_rest_ecg_sign = RestingECG,
                                    num_max_heartrate_bpm = MaxHR,
                                    bin_exercise_induced_angina = ExerciseAngina,
                                    num_oldpeak_in_st_depression = Oldpeak,
                                    fac_st_slope = ST_Slope,
                                    bin_heart_disease_label = HeartDisease
                                    )
```

Next we want to go through and see how we handle **specific recoding of data types** to our future work easier.

Example - recoding text columns into a binary factor Note the use here of two useful **dplyr** functions, **mutate** which directly manipulates data frames and **recode_factor** which allows you to write and show your operation in a clearly understandable way.

```
recodedHeartFailure %<>% dplyr::mutate(bin_sex_is_male=dplyr::recode_factor(bin_sex_is_male,
                         'M' = '1',
                         'F' = '0'))
```

In this code fragment we also meet the command "%<>%", which simply means perform the commands on the right hand side and feed the output back into the object on the left hand side. Watch how this simple trick makes your code a bit shorter to write and change (if needed), by reducing duplicate text.

```
recodedHeartFailure <- dplyr::mutate(recodedHeartFailure,
                                    bin_sex_is_male=dplyr::recode_factor(bin_sex_is_male,
                         'M' = '1',
                         'F' = '0'))
```

```
recodedHeartFailure$bin_fasting_blood_gluc_gt_120mg_per_dl %<>% as.factor
```

Example - numerical column of 1 and 0 recoded explicitly as binary

```
recodedHeartFailure %<>% dplyr::mutate(fac_rest_ecg_sign=dplyr::recode_factor(fac_rest_ecg_sign,
                                                                              'Normal' = 'N',
                                                                              'ST' = 'ST',
                                                                              'LVH' = 'LVH',
                                                                              .default = NULL))
```

Example - defining a catch all category when you recode

Recoding data in a single code block - while doing it the clean and highly visible way

```
rawHeartFailure <- read.csv(file.path(pathToData,'Heart failure.csv'))

recodedHeartFailure <- dplyr::rename(rawHeartFailure,
                                     num_age_in_years = Age,
                                     bin_sex_is_male = Sex,
                                     fac_chest_pain_type = ChestPainType,
                                     num_rest_blood_press_mmHg = RestingBP,
                                     num_serum_cholestrol_mm_per_dl = Cholesterol,
                                     bin_fasting_blood_gluc_gt_120mg_per_dl = FastingBS,
                                     fac_rest_ecg_sign = RestingECG,
                                     num_max_heartrate_bpm = MaxHR,
                                     bin_exercise_induced_angina = ExerciseAngina,
                                     num_oldpeak_in_st_depression = Oldpeak,
                                     fac_st_slope = ST_Slope,
                                     bin_heart_disease_label = HeartDisease
                                     )

recodedHeartFailure <- dplyr::mutate(recodedHeartFailure,
    #
    num_age_in_years = as.numeric(num_age_in_years),
    #
    bin_sex_is_male=dplyr::recode_factor(bin_sex_is_male,
                         'M' = '1',
                         'F' = '0'),
    #
    fac_chest_pain_type = as.factor(fac_chest_pain_type),
    #
    num_rest_blood_press_mmHg = as.numeric(num_rest_blood_press_mmHg),
    #
    num_serum_cholestrol_mm_per_dl = as.numeric(num_serum_cholestrol_mm_per_dl),
    #
    bin_fasting_blood_gluc_gt_120mg_per_dl = as.factor(bin_fasting_blood_gluc_gt_120mg_per_dl),
    #
    fac_rest_ecg_sign=dplyr::recode_factor(fac_rest_ecg_sign,
                                           'Normal' = 'N',
                                           'ST' = 'ST',
                                           'LVH' = 'LVH',
                                           .default = NULL),
    #
    num_max_heartrate_bpm = as.numeric(num_max_heartrate_bpm),
    #
    bin_exercise_induced_angina=dplyr::recode_factor(bin_exercise_induced_angina,
                                                     'N' = '0',
                                                     'Y' = '1'),
    #
    num_oldpeak_in_st_depression = as.numeric(num_oldpeak_in_st_depression),
    #
    fac_st_slope=dplyr::recode_factor(fac_st_slope,
                                      'Up' = '1',
                                      'Flat' = '0',
                                      'Down' = '-1'),
    #
    bin_heart_disease_label = as.factor(bin_heart_disease_label)
```

```
  )
```

Here is the summary command on the raw imported data.

```
summary(rawHeartFailure)
```

```
##       Age            Sex            ChestPainType        RestingBP
##  Min.   :28.00   Length:918         Length:918         Min.   :  0.0
##  1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0
##  Median :54.00   Mode  :character   Mode  :character   Median :130.0
##  Mean   :53.51                                         Mean   :132.4
##  3rd Qu.:60.00                                         3rd Qu.:140.0
##  Max.   :77.00                                         Max.   :200.0
##   Cholesterol      FastingBS       RestingECG            MaxHR
##  Min.   :  0.0   Min.   :0.0000   Length:918         Min.   : 60.0
##  1st Qu.:173.2   1st Qu.:0.0000   Class :character   1st Qu.:120.0
##  Median :223.0   Median :0.0000   Mode  :character   Median :138.0
##  Mean   :198.8   Mean   :0.2331                      Mean   :136.8
##  3rd Qu.:267.0   3rd Qu.:0.0000                      3rd Qu.:156.0
##  Max.   :603.0   Max.   :1.0000                      Max.   :202.0
##  ExerciseAngina        Oldpeak          ST_Slope          HeartDisease
##  Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000
##  Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000
##  Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000
##                     Mean   : 0.8874                      Mean   :0.5534
##                     3rd Qu.: 1.5000                      3rd Qu.:1.0000
##                     Max.   : 6.2000                      Max.   :1.0000
```

And here is the same summary command on the raw imported data.

```
summary(recodedHeartFailure)
```

```
##  num_age_in_years bin_sex_is_male fac_chest_pain_type num_rest_blood_press_mmHg
##  Min.   :28.00    1:725           ASY:496             Min.   :  0.0
##  1st Qu.:47.00    0:193           ATA:173             1st Qu.:120.0
##  Median :54.00                    NAP:203             Median :130.0
##  Mean   :53.51                    TA : 46             Mean   :132.4
##  3rd Qu.:60.00                                        3rd Qu.:140.0
##  Max.   :77.00                                        Max.   :200.0
##  num_serum_cholestrol_mm_per_dl bin_fasting_blood_gluc_gt_120mg_per_dl
##  Min.   :  0.0                  0:704
##  1st Qu.:173.2                  1:214
##  Median :223.0
##  Mean   :198.8
##  3rd Qu.:267.0
##  Max.   :603.0
##  fac_rest_ecg_sign num_max_heartrate_bpm bin_exercise_induced_angina
##  N  :552           Min.   : 60.0         0:547
##  ST :178           1st Qu.:120.0         1:371
##  LVH:188           Median :138.0
##                    Mean   :136.8
##                    3rd Qu.:156.0
```

```
##                   Max.   :202.0
##  num_oldpeak_in_st_depression fac_st_slope bin_heart_disease_label
##  Min.   :-2.6000              1 :395       0:410
##  1st Qu.: 0.0000              0 :460       1:508
##  Median : 0.6000              -1: 63
##  Mean   : 0.8874
##  3rd Qu.: 1.5000
##  Max.   : 6.2000
```