**Doctors as Data Scientists Session 2 - 3**

Login to R studio cloud (now called posit cloud) and click on the

**Doctors as Data scientists** workspace

Find the project “Session 3” and click “Start”.

Chart, scatter chart

Description automatically generated

check you can see a series of files in the bottom right hand “files” part of the window. These files should include .csv files such as “observations….” “Medical\_dictionary….”, “Patients….” etc etc

**SAVING YOUR WORK TO AN R SCRIPT FILE**

Before we start, either open your previous R script file, or make a new one, so that you can save and edit your work easily.

To save your work, go to the bottom right of your screen and click “new blank file” and select “R script” as below:

Graphical user interface, text

Description automatically generated

Save the file as something meaningful – I suggest “yourname\_Doctors\_As\_DataScientists” or similar. You will then see a new window pop up as below:

Graphical user interface, text, application, email

Description automatically generated

From now on, type your commands into the R script window at the top, and hit the “run” button to see the results come out in the console window below.

*IMPORTANT NOTE – ONCE YOU HAVE TYPED A COMMAND INTO YOUR R SCRIPT FILE YOU NEED TO PRESS THE “RUN” BUTTON TO MAKE IT WORK AND PRODUCE RESULTS IN THE CONSOLE BELOW. PRESSING RETURN DOES NOT MAKE THE CODE RUN EVEN THOUGH IT MOVES YOU ONTO THE NEXT LINE IN YOUR R SCRIPT FILE*

**EXERCISE 5 IMPORTING AND WORKING WITH A DATA FILE AND CHECKING IT**

Lets work with some data. In this session we will learn 3 things:

1. Importing a dataset – in R the terminology is a “dataframe”. We will use a subset of primary care data from the CPRD (Clinical practice research datalink) resource: <https://cprd.com/> . except that this data is not from real patients - it looks exactly like that from the real primary resource that has >10 million patients in, but is “synthetic” each row is a fake patient, so that we do not share real patient records, but the data looks very real. You can read more about this resource and even apply to use it yourself [here](https://cprd.com/synthetic-data#CPRD%20cardiovascular%20disease%20synthetic%20dataset).
2. Taking a look at the data so we can understand it.
3. Subsetting the data – selecting particular cells, columns and rows to make and look at a smaller version of the data – this is part of what is called **“Data wrangling”** – getting your data sorted out before analysing it. This is really important when data includes information from millions of patients and 100 millions of observations and prescriptions.

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: IMPORTING IN A DATASET**

First you will need to load in the tidyverse library with

library(tidyverse)

Loading the tidyverse library provides R Studio with more background information and ability to run a wider range of commands. You need to load the tidyverse library each time you open R studio.

First we will import a small test dataset to get used to the functions and data formats, before looking at the larger CPRD data (more on CPRD here: <https://cprd.com/> )

We will do learn and do several things with a small test data set:

1. Import it into your working environment with the **Read.csv()** function
2. Describe it, using the **Head(), Head(, n= X), Glimpse(), View()** and other similar functions
3. Learn the format of the values in each column such as **int, dbl, chr**
4. The **[,]** concept as the coordinates of a matrix – important for subsetting a dataset – selecting a subset of columns and rows.
5. the importance of removing missing data for calculations **na.rm = TRUE,** where na means “not available” and rm means “remove”.
6. Some alternatives to the **[,]** approach to subsetting, including the **$** sign for selecting the name of a column.

First , use the read.csv() function to read in the test dataset, assigning with the <- assigning operator the data to an R object in your environment, called “testdata”:

testdata <- read.csv("Test\_data\_patients.csv")

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: DESCRIBING A DATASET**

now take a look at it with a variety of functions and discuss what each result means

dim(testdata)

nrow(testdata)

ncol(testdata)

ls(testdata)

head(testdata)

head(testdata, n= 10)

after this head() function what is the additional n = 10 part doing ?

glimpse(testdata)

after this glimpse() function , what do you think the int, chr and dbl mean ?

view(testdata)

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: SUBSETTING A DATASET**

Next , by placing numbers between 1 and 10 in the square brackets can you work out what the following operations are doing ?.

Note the syntax for this important use of square brackets is

Mydata[row coordinate, column coordinate] with the important additional function that leaving one of these two coordinates blank selects all

testdata\_subset <- testdata[1,3]

testdata\_subset <- testdata[2,3]

testdata\_subset <- testdata[,3]

testdata\_subset <- testdata[,4]

testdata\_subset <- testdata[,1:3]

testdata\_subset <- testdata[,c(1:3,6)]

What does the final command above do, what is the c() function within the square brackets doing ?

Next, what are the following functions doing ? did both return the minimum value ?:

min(testdata [,3])

min(testdata[,3], na.rm = TRUE)

Next, what are the following functions doing ? did both return the mean value ?:

mean(testdata [,3])

mean(testdata[,3], na.rm = TRUE)

**QUESTION – WHAT DOES THE FOLLOWING FUNCTION DO AND WHY WOULD IT NOT MAKE MUCH SENSE TO RUN IT !?**

Testdata\_subset <- [,]

IF YOU ARE NOT SURE, RUN IT AND CHECK IT, WITH DIM() AND OTHER DESCRIPTIVE FUNCTIONS.

Now we have the concept of choosing the coordinates, and just using a space if we wish to select all rows or columns , NOTE WE CAN USE SHORTCUTS.

Try the following:

mean(testdata [,”height”], na.rm = TRUE)

mean(testdata [,”weight”], na.rm = TRUE)

We can also use the select() function as a shortcut, to select only certain columns into a subset. The select() function has the format

select(mydata, “column x”, “column y”, “column z”)

Try selecting certain columns and assigning the data to a new dataframe named “testdata\_subset”:

testdata\_subset <- select(testdata, "PatientID", "gender", "height", "weight")

view(testdata\_subset)

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: SUBSETTING A DATASET, FILTERING ON VALUES WITHIN COLUMNS NOT JUST THE COLUMNS:**

what if you wanted to look at the min, max and mean weights and heights separately by sex

**create a new dataframe, that only includes women, using the filter() function using the syntax: n**ew\_dataframe <- filter(current dataframe, column on which to select == "values in column you #want to select")

testdata\_women <- filter(testdata, gender == "2")

and repeat your commands but making sure to refer to the new dataframe:

mean(testdata\_women [,3], na.rm = TRUE)

**QUESTION – TO CHECK YOUR UNDERSTANDING, CREATE A SUBSET OF THE DATA THAT CONSISTS ONLY OF MEN BORN BEFORE 1983. TIP USE THE & OPERATOR TO SELECT TWO VARIABLES ON WHICH YOU WISH TO FILTER AT THE SAME TIME.**

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: joining**

To do this exercise, import a second file into a dataframe– the Medical dictionary

Meddictionary <- read.csv("Med\_dictionary\_for\_students.csv")

Check what it looks like with some descriptive commands, how many medical terms does it have ?

head(Meddictionary)

nrows(Meddictionary)

etc

Now do the joining, and view the results:

Testdata\_innerjoined <- inner\_join (testdata, Meddictionary, by = "medcodeid")

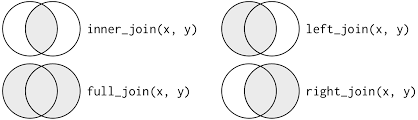
view(testdata\_joined)

and try some different types of join (DO NOT try right\_join() or full\_join with the below – why not ?)

testdata\_leftjoined <- left\_join (testdata, Meddictionary, by = "medcodeid")

view(testdata\_leftjoined)

here is a useful visual of how joining works:



**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: selecting based on search terms**

Now use a pattern search to make a subset of the Medical dictionary with only the term “cardiac” in. For this you need to use the additional function data.frame() to tell R to place the results into a dataframe. You then use the str\_subset() function to search for a pattern.

Meddictionary\_cardiac <- data.frame(str\_subset(Meddictionary [,"Term"], pattern = "cardiac"))

nrows(Meddictionary\_cardiac)

Now use a pattern search together with the str\_subset() function to search for another pattern you think will be in the medical dictionary.

**EXERCISE 5.0 PRACTICE RUN WITH A SMALL DATASET: JOINING TO ANOTHER SUBSET**

**SESSION 3:**

**NEXT WE WILL PUT INTO PRACTICE WHAT WE HAVE LEARNT WITH THE VERY SMALL DATASET, BUT USING THE MUCH LARGER CPRD PRIMARY CARE DATA:**

Now use the read() function to “read in” data from the CPRD files into your working environment. Note two things:

1. The file “Patient.csv” exists in your folder, but you cannot use it until you read it into your working environment.
2. a set of data can be thought of as a matrix, (it has rows and columns) but is better to think of as a “data frame”. Data frames and matrices are the same – rows and columns of information, except that data frames can include non numerical information, such as words like “male” and “female”, “yes” and “no”.

To use the read() function we need to tell the computer where the file with the data is.

The read() function allows us to move the file into your working environment. The syntax is

mydataframe <- read("myfile ", header = TRUE)

but we might need to use read.csv as the file is a .csv file, so

try

patients\_dataframe <- read.csv("Patient.csv", header = TRUE)

**EXERCISE 5.2 TAKING A LOOK AT THE DATA**

Now you’ve imported part of the large CPRD dataset into the the dataframe patients\_dataframe it is important to do some checks. Here are several new functions to take a brief look at the data. Now that we have 1000s of people we cant really look at the whole lot.

First lets find out the size of the dataset to find out its basic size.

dim(patients\_dataframe)

nrow(patients\_dataframe)

ncol(patients\_dataframe)

Try the list ls() function to obtain some more information:

ls(patients\_dataframe)

And the head() function

head(patients\_dataframe)

You can see that head() function produces just a few rows of information at the top (the head) of the dataframe. This allows you to get a feel for the number of columns, what they are called and what is in them.

But the head() function only produces the first 6 rows. What if I’d like to take a look at a few more ?

Try using an argument, after the comma to change number of rows displayed :

head(patients\_dataframe, n = 10)

similarly , you may want to look at the end of the dataframe, so try

tail(patients\_dataframe)

tail(patients\_dataframe, n = 10)

another useful command is glimpse, try

glimpse(patients\_dataframe)

and, to really see the data in the nicest format, try

view(patients\_dataframe)

Note how this brings up the data in a new window and looks much like a spreadsheet.

Now use the read.csv() function and assignment operator <- to read in two other files that are in the folder and take a look at the data. NOTE you will need to change the names of the files to load in , and the names of the dataframes into which you wish to load them.

Load the following files into new dataframes using the read.csv() function and <- assignment operator:

Med\_dictionary\_for\_students.csv

And

observations\_for\_students.csv

you should now have 3 dataframes in your R studio environment in the top right:

observations\_dataframe

patients\_dataframe

Med\_dictionary\_dataframe

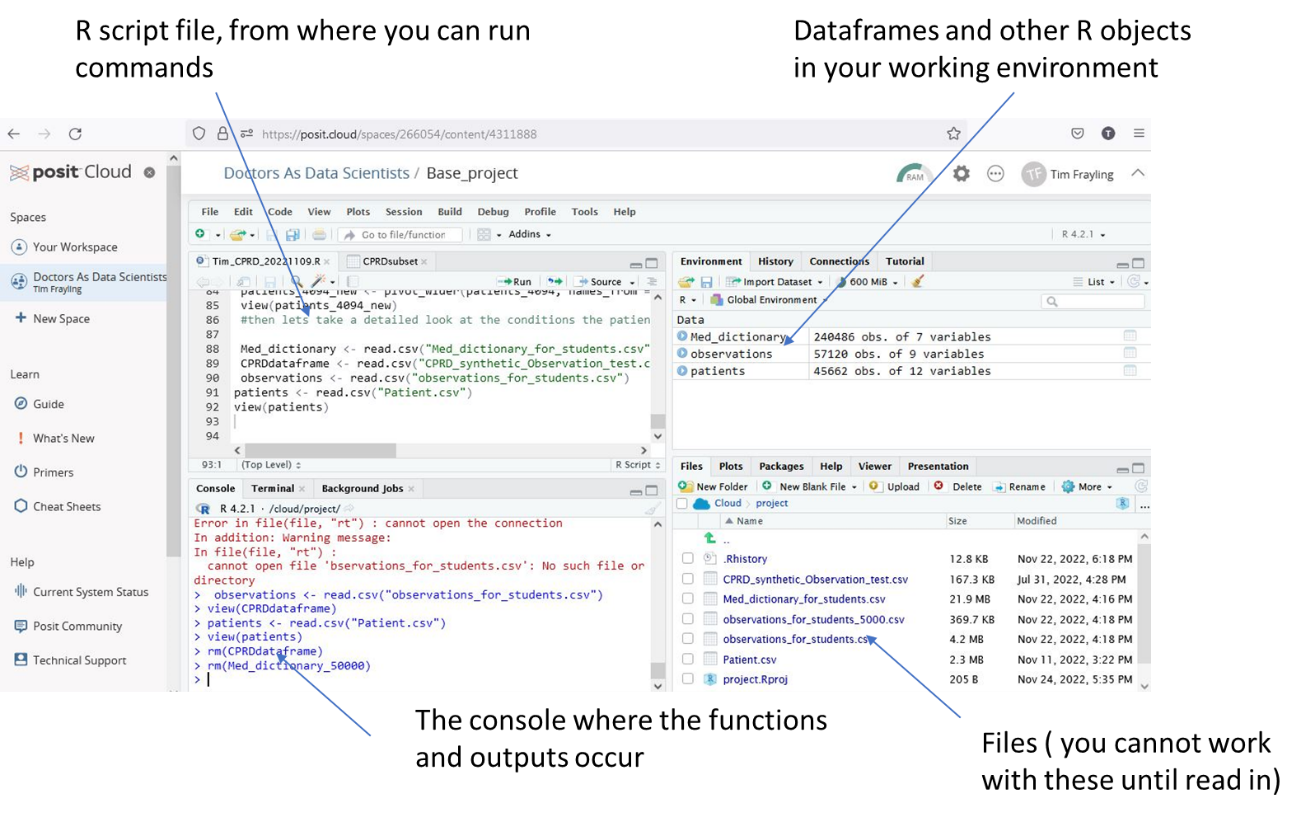
If you do not have these three dataframes, check you read the files into dataframes with commands such as these:

observations\_dataframe <- read.csv("observations\_for\_students.csv", header = TRUE)

Med\_dictionary\_dataframe <- read.csv("Med\_dictionary\_for\_students.csv

", header = TRUE)

If you did not name your dataframes with these exact names, it is not a problem but you will have to use your names for all the further commands below:



Check the other two dataframes with the set of ls(), glimpse(), head(), nrow(), view() etc commands

BACKUP PLAN NOTE there are smaller versions of the medical dictionary and observations dataframes in case the large number of students crashes our space on R studio cloud.

observations\_for\_students\_5000

Med\_dictionary\_for\_students\_small

**EXERCISE 5.3 INTRODUCTION TO “DATA WRANGLING” SUBSETTING DATA**

Now lets be a little more selective and check specific cells, columns and rows of data in more detail Lets look at the “yob” variable, which is the fifth column of the patients dataframe. To do this we use the **square brackets**, which are key to subsetting.

The values either side of the comma in the square brackets represent the coordinates of a matrix – in our case, the dataframe.

The format is

mydataframe[row coordinate,column coordinate]

#so the following command selects the value in the first row and fifth column:

patients\_dataframe [1,5]

#The following command selects the value in the second row and fifth column:

patients\_dataframe [2,5]

#The following command selects the value in the second row and sixth column:

patients\_dataframe [2,6]

#select the value in the fifth column of the third row, and provide the class (type) of data that value is using the class() function

class(patients\_dataframe [3,5])

#Selecting individual values may not be that helpful, but we can easily select all rows for selected #columns using

patients\_dataframe [,5]

#The lack of a value before the comma means “all” – so because it is before the comma – this means #“all rows”. This can be useful as we can now do some summarising of particular columns of data.

#For example, using the min() and max() functions, try

min(patients\_dataframe [,5])

max(patients\_dataframe [,5])

#select the fifth column for all the rows, and provide the mean

mean(patients\_dataframe [,5])

#next try the following ,

mean(patients\_dataframe [,5], na.rm = TRUE)

#na.rm means “remove NAs” – NA meaning not available, so missing data point

#try also plotting the data to get a feel for what the data looks like For example, create a chart of the #Practice IDs, to see which GP Practices have most patients in the data.

hist(patients\_dataframe [,5])

#but it might be annoying to count up which column is in which position. There are actually several #ways of selecting particular columns in the data:

#try the histogram function hist() for the yob variable in these three ways:

The following means – select all rows and the 5th column from patients\_dataframe and plot a histogram of the data:

hist(patients\_dataframe [,5])

The following means – select all rows and the column headed “yob” from patients\_dataframe and plot a histogram of the data:

hist(patients\_dataframe [,”yob”])

The following means – select all rows and the column headed “yob” from patients\_dataframe and plot a histogram of the data (i.e. the same as above – the $ sign is a handy short cut to using the square brackets to select columns of data:

hist(patients\_dataframe$yob)

#Try another descriptive function to confirm these three ways all produce the same result – to find #the earliest year of birth

min(patients\_dataframe [,5])

min(patients\_dataframe [,”yob”])

min(patients\_dataframe$yob)

#and to find the latest year of birth, confirm these commands all do the same thing:

max(patients\_dataframe [,5])

max(patients\_dataframe [,”yob”])

max(patients\_dataframe$yob)

#now try similar functions to summarize the observation types in the observation dataframe e.g.

hist(observations\_dataframe$obstypeid)

#try the tabulate function to return the total numbers of each observation type

tabulate(observations\_dataframe$obstypeid)

#to find out more about the obstype, read in the small ObsType.csv file with the read.csv() function

Observations\_type <- read.csv(“ObsType.csv”)

And use glimpse() or head() to take a look

#now tabulate some of the categorical columns – those with only a few values, such as gender, try, #in patient dataframe

tabulate(patients\_dataframe[,”gender”])

#to find out which way round men and women are coded, and if there are more than 2 categories, #take a look at the very small Gender.csv file that summarizes the information. You will need to use #the read.csv() function again to read in the data into a small dataframe.

**#NOTE TO PROVIDER – STOP AND RECAP CONCEPT OF [,] AS COORDINATES IN A MATRIX**

**QUESTION 1** USING THE PATIENTS DATAFRAME, SQUARE BRACKETS , THE <- ASSIGNMENT OPERATOR, AND THE MIN() AND MAX() FUNCTIONS, PLACE THE EARLIEST YEAR OF BIRTH IN A VECTOR *earliest\_yob*

USE THE PASTE() FUNCTION INSIDE THE PRINT() FUNCTION TO PRINT A STATEMENT WITH THE RESULT THAT LOOKS LIKE THE FOLLOWING, BUT WITH THE ACTUAL YEAR OF BIRTH REPLACING “*THE EARLIEST YOB”).* REMEMBER TO USE A COMMA TO SEPARATE THE ELEMENTS YOU WISH TO PASTE TOGETHER BEFORE PRINTING

THE EARLIEST YEAR OF BIRTH IN THE GP DATABASE IS *THE EARLIEST YOB*

**QUESTION 2** USING THE PATIENTS DATAFRAME, SQUARE BRACKETS , THE <- ASSIGNMENT OPERATOR, AND THE MAX() FUNCTION, PLACE THE LATEST YEAR OF BIRTH IN A VECTOR *Latest\_yob*

USE THE PASTE() FUNCTION INSIDE THE PRINT() FUNCTION TO PRINT A STATEMENT WITH THE RESULT THAT LOOKS LIKE THE FOLLOWING, BUT WITH THE ACTUAL YEAR OF BIRTH REPLACING “*THE LATEST YOB”).* REMEMBER TO USE A COMMA TO SEPARATE THE ELEMENTS YOU WISH TO PASTE TOGETHER BEFORE PRINTING

YES THERE ARE CHILDREN IN THE DATASET BECAUSE THE LATEST YEAR OF BIRTH IN THE GP DATABASE IS *THE EARLIEST YOB*

**QUESTION 3.** NOW WRITE AN IF ELSE STATEMENT TO REPORT WHETHER OR NOT THE PATIENTS INCLUDE CHILDREN

REMEMBER THE IF ELSE SYNTAX

IF (SOMETHING IS TRUE) { DO SOMETHING E.G. PRINT A MESSAGE}

ELSE {DO SOMETHING ELSE E.G. PRINT AN ALTERNATIVE MESSAGE}

IN THE () BRACKETS YOU COULD TEST WHETHER OR NOT THERE ARE CHILDREN PRESENT WITH latest\_yob > 2004

REMEMBER TO PRINT A MESSAGE IN ENGLISH YOU NEED SPEECH MARKS E.G. “YES THERE ARE CHILDREN IN THE PATIENT DATABASE”

**EXERCISE 5.4 MORE DETAILED SUBSETTING AND ORGANISING OF DATA – “DATA WRANGLING”**

Now lets do some more detailed subsetting and data wrangling, this time with another part of the CPRD database:

Now we will create a subset of the data. Lets say we want to create a dataframe with only certain columns and rows in it.

For example, lets imagine we want to look only at females from one particular GP practice and we are only interested in certain columns of information.

We will break this problem down. First lets select some columns and place them in a new dataframe, you can use a colon : to select a set of contiguous columns:

#To select all rows (in this case each row represents a patient) but only columns 1, 2 and 3, and place #them in a new dataframe try:

patients\_subset <- patients\_dataframe[,1:3]

#Take a closer look at the Patients\_subset dataframe with the glimpse() function and see if there are #columns that don’t look useful.

glimpse(patients\_subset)

#you will see there is lots of missing data in the mob (month of birth), emisddate and cprdddate #columns – as the values are either “NA” – meaning not available - or “”.

#So lets make a smaller dataset, that will work faster, by selecting all the other columns. There are #several ways of doing this. You can select the columns you want using the concatenate c() function, #within the square brackets

#So the following means concatenate columns 1 to 5 with columns 8 to 11, the comma separating #the variables you wish to concatenate.

#note the c() function means combine / concatenate together:

#c(1:5,7:11)

#and the following means, for all rows in the dataframe patients, combine columns 1 to 5 with #columns 8 to 11

#patients[,c(1:5,8:11)]

#And now try the full operation, assigning the results to a new subset of the data

patients\_subset <- patients\_dataframe[,c(1:3,8:10)]

#And

glimpse(patients\_subset)

#and to take a look at the new subset of data

view(patients\_subset)

# try and make a few alternative subsets of the data, including all rows, but only certain columns. #Note that you can use a minus symbol to indicate columns you don’t want to include

#For example, using the assignment operator and square brackets make subsets of the data #including all the columns except columns 6 and 7 which contain month of birth, and emis date.

patients\_subset <- patients\_dataframe[,-(6:7)]

#Use glimpse() , head() and, ncol() etc to check out if this has worked

#NOTE emis is one of the GP software providers. More here if you are interested: #<https://www.emishealth.com/products/emis-web>

#Next to start making a subset, first select women only, with the filter() command. The filter #command has the following format

#New\_dataframe <- filter(current dataframe, column on which to select == "values in column you #want to select")

#So to select all women

patients\_female <- filter(patients\_dataframe, gender == "2")

#Select one practice only – lets select the practice with fewest patients, try

tabulate(patients\_female[,”pracid”])

**#**to tabulate the practice ID column and work out which practice has #the fewest patients, then:

patients\_female\_smallest\_practice <- filter(patients\_female, pracid == "2")

#And then only the female patients from the smallest practice and who are adults (born before #2004):

patients\_adultfemale\_smallest\_practice <- filter(patients\_female\_smallest\_practice, yob < 2004)

**QUESTION 4.1.** USE THE & OPERATOR WITHIN THE FILTER() FUNCTION TO ADAPT THE COMMANDS YOU HAVE JUST USED TO SELECT THE ADULT WOMEN FROM PRACTICE 2 ALL IN ONE GO. CHECK EVERYTHING HAS WORKED WELL WITH SOME DESCRIPTIVE FUNCTIONS

**QUESTION 4.2.** USE THE NROW() FUNCTION WITHIN THE PASTE() FUNCTION WITHIN THE PRINT() FUNCTION, TOGETHER WITH COMMAS, TO PRINT OUT THE NUMBER OR ADULT FEMALE PATIENTS IN GP PRACTICE 2 (THE SMALLEST ONE)

**QUESTION 5.** MAKE A SUBSET OF THE DATA FEATURING ONLY MEN FROM THE PRACTICE WITH THE MOST PATIENTS AND BORN BEFORE 1970. RATHER THAN PERFORMING THE SUBSETTING IN THREE STEPS CAN YOU WRITE A COMMAND THAT USED THE “&” OPERATOR TO DO IT ALL IN ONE STEP ?

YOU CAN MAKE THIS MUCH EASIER BY COPYING AND PASTING THE COMMANDS FROM QUESTION 4 AND CHANGING ONLY THE KEY VARIABLES AND TEXT.

**EXERCISE 5.5 SOME MORE SUBSETTING**

**A reminder - we are using 3 dataframes – the details of the patients, their observations and the medical dictionary. Lets filter them to make them smaller and more manageable.**

#Imagine we are interested only in one GP practice - use the filter() function to subset the patients #to include only patients from practice ID 13

Patients\_prac13 <- filter(patients\_dataframe, pracid == "13")

#and make a smaller version of the Medical dictionary file and rename some column headings

#first rename

#note the format is rename(dataframe, newname = oldname)

Med\_dictionary\_dataframe <- rename(Med\_dictionary\_dataframe, Medical\_term = Term)

#check that has worked

glimpse(Med\_dictionary\_dataframe)

view(Med\_dictionary\_dataframe)

**QUESTION 6** – CAN YOU USE SIMILAR COMMANDS TO RENAME THE GENDER COLUMN IN THE PATIENTS DATAFRAME AS SEX ?

#Now make a smaller version of the Medical dictionary file and rename some column headings

# we can do this all in one go using a new function select() . We can select only some of the columns, and rename at the same time :

#the syntax of the select() function is:

#select (dataframe, columnx, columny, columnz, etc) where columns x y and z are the names of the #columns you want

#try subsetting the Med\_dictionary dataframe so that it only includes the medcodeid, Medical\_Term #and SnomedCTConceptId coluumns

#try this in two different ways

#first using the square brackets :

Med\_dictionary\_version1 <- Med\_dictionary\_dataframe[,c(2,3,5)]

#or using the select() function

Med\_dictionary\_version2 <- select(Med\_dictionary\_dataframe, medcodeid, Medical\_term, SnomedCTConceptId)

#view the two new versions of the Med\_dictionary, you should find they look the same.

**EXERCISE 5.6 SUBSETTING ON NATURAL LANGUAGE SEARCH TERMS**

#use a new function str\_subset() to search the text in the "Medical\_Term" column to see which #types of conditions are present in the patients

str\_subset(Med\_dictionary\_version1 [,"Medical\_term"], pattern = "cardiac")

#now place the summary of cardiac conditions into its own dataframe to make it easily viewable:

cardiac\_conditions <- data.frame(str\_subset(Med\_dictionary\_version1 [,"Medical\_term"], pattern = "cardiac"))

view(cardiac\_conditions)

#how many different terms contain "cardiac"

#try a medical term you think might be rarer

congenital\_conditions <- data.frame(str\_subset(Med\_dictionary\_version1 [,"Medical\_term"], pattern = "congenital"))

view(congenital\_conditions)

glimpse(cardiac\_conditions)

**#EXERCISE 5.7 JOINING**

#next lets join up the medical\_dictionary dataframe, to the patient observations dataframe

#join the observations dataframe to the medical dictionary

#to do this make a new dataframe called observations\_medcode or choose a name yourself

observations\_medcodes <- inner\_join (observations, Med\_dictionary\_version1, by = "medcodeid")

#take a look at the data

glimpse(observations\_medcodes)

#now you have the patients with their IDs joined to the medical terms.

#take a look at a patient by their ID, to see how many medical terms they have.

#now you have the patients with their IDs joined to the medical terms.

#take a look at a patient by their ID, to see how many medical terms they have.

patient\_111669000000\_medterms <- select(observations\_medcodes, patid, medcodeid, Medical\_term)

#note the use of the unique() function because joining datasets often results in duplication of rows

patient\_111669000000\_medterms <- unique(filter(patient\_111669000000\_medterms, patid == 111669000000))

view(patient\_111669000000\_medterms)