|  |
| --- |
| NIHR SPCR Seed Corn Funding 2023/2024 |
| Training Materials |
| An Introduction to using CPRD Aurum |

|  |
| --- |
| Institutional affiliation: Keele University  © University of Keele 2024  **This work is licensed under Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/4.0/** |

Written by Noor Daud

Contact: k.n.daud@keele.ac.uk

Table of Contents

[1. Introduction 3](#_Toc174015918)

[1.1 Setting up the CPRD Aurum files 3](#_Toc174015919)

[1.2 Saving CPRD files 4](#_Toc174015920)

[1.3 Stata coding 4](#_Toc174015921)

[1.4 How to use the training materials 5](#_Toc174015922)

[2. Protocol 7](#_Toc174015923)

[Tracking document: 8](#_Toc174015924)

[3. Code lists 9](#_Toc174015925)

[3.1 What are code lists? 9](#_Toc174015926)

[3.2 Where do I find code lists? 9](#_Toc174015927)

[3.3 How do I produce code lists? 10](#_Toc174015928)

[**Example: Creating a Gabapentinoid code list.** 11](#_Toc174015929)

[4. Extracting observations and prescriptions using code lists 16](#_Toc174015930)

[**Example 1: Extracting the Gabapentinoid codelist.** 16](#_Toc174015931)

[**Example 2: Extracting an Osteoporosis codelist.** 18](#_Toc174015932)

[5. Study population 21](#_Toc174015933)

[5.1 Patient file 21](#_Toc174015934)

[5.2 Practice file 22](#_Toc174015935)

[6. Units 25](#_Toc174015936)

[**Example: eGFR Units with restriction** 25](#_Toc174015937)

[General metric conversions: 27](#_Toc174015938)

[**Exercise 1** 28](#_Toc174015939)

[7. Defining other variables 30](#_Toc174015940)

[**Exercise 2** 32](#_Toc174015941)

[8. HES APC 33](#_Toc174015942)

[8.1 ICD-10 Codelist 33](#_Toc174015943)

[Synonym searching: 33](#_Toc174015944)

[**Example: Searching for fracture in the NHS ICD-10 5th Edition data files** 36](#_Toc174015945)

[8.2 OPCS Codelist 38](#_Toc174015946)

[**Example: Searching for fracture in the NHS OPCS 4 data files** 38](#_Toc174015947)

[8.3 Patient file HES APC 40](#_Toc174015948)

[8.4 Extracting ICD-10 codelist from HES APC 42](#_Toc174015949)

[**Example 1: Extracting fractures ICD-10 codelist from Primary HES diagnoses.** 42](#_Toc174015950)

[**Example 2: Extracting fractures ICD-10 codelist from HES diagnoses hospitalisation.** 44](#_Toc174015951)

[**Example 3: Extracting fractures ICD-10 codelist from HES diagnoses episode.** 46](#_Toc174015952)

[8.5 Extracting OPCS codelist from HES APC 49](#_Toc174015953)

[**Example: Extracting fractures opcs codelist.** 49](#_Toc174015954)

[9. Solutions 52](#_Toc174015955)

[10. Final thoughts 54](#_Toc174015956)

[References 55](#_Toc174015957)

# Introduction

These training materials were started to help those that are unfamiliar or want a refresher on using CPRD (Clinical Practice Research Datalink) Aurum data. When starting CPRD Aurum work, I was not sure where or how to start – these materials give a step-by-step guide on how to approach working with CPRD Aurum data, written from my experience of using CPRD data and help from the Electronic Health Record group at Keele University.

To begin with, it is beneficial to familiarise yourself with what CPRD is. This can be found: [Clinical Practice Research Datalink | CPRD](https://cprd.com/). There are informational resource modules in the right-hand side of the tab bar and covers both Aurum and Gold. The ‘Denominator data’ resource module shows screenshots of what the data will look like and how to use denominator data.

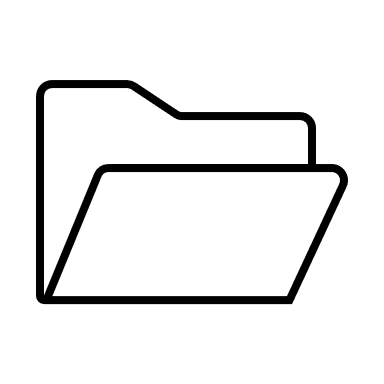
This document is supposed to be an aid to complement CPRD’s own resources when working with CPRD Aurum data. We will go through the steps needed and some example code will be provided. Note that example code is generated from Stata version 18.0.

In addition to these training materials, informal training may also be available from your relevant Data manager. For more information, please contact your Data Manager or CPRD data key fob holder.

## Setting up the CPRD Aurum files

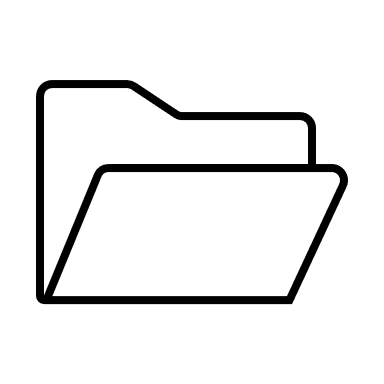
The Aurum data is downloaded and saved onto a secure folder by the data manager. For more information on saving CPRD data, see Chapter 1.2. The data is then separated into multiple files which may contain .txt files. I usually add an extra 3 folders for ease when working with CPRD data as it can help to keep things separated and organised.

These are the folders I add:



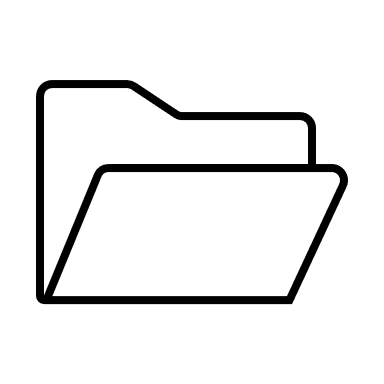
Codes

This folder will include codelists and log files relating to codelists. I usually add extra folders within this to separate the codelist from the logs, and finalised revised codelists from the initial codelists. It is convenient to separate the finalised codelists from the initial codelists as you can then upload these onto a repositry if needed.



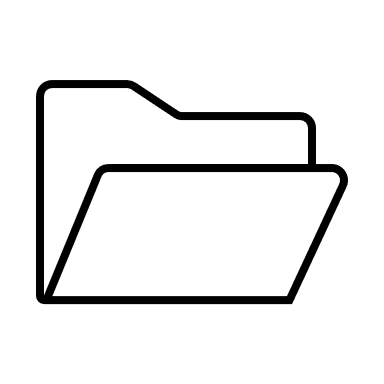
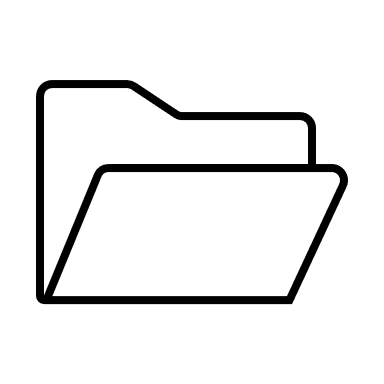
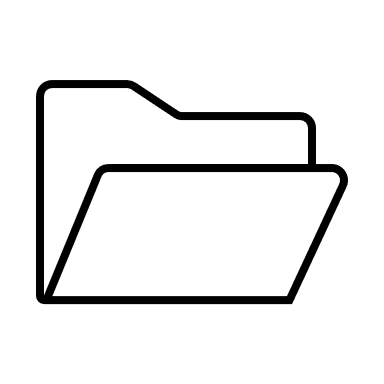
eRAP

This folder contains the most up to date eRAP form, I include this as it is easy to refer to when using the CPRD data. I also include the ‘Tracking’ excel document here (Chapter 2).



Stata files

This folder includes your main Do-Files (defining population, applying algorithms, analysis… etc), Help files (do-files that others have written that you can use as reference) and a Working folder which includes ready-to-use CPRD Aurum .dta files. I usually add extra folders within the Stata files folder:



Do-files

Help files

Working

## Saving CPRD files

All files relating to your CPRD work **MUST** be stored onto a secure folder. If you have any questions regarding saving CPRD material, contact your Data Manager or CPRD data key fob holder.

## Stata coding

These training materials require previous experience of using Stata, as we cover loops and intermediate Stata commands. Don’t worry if you have limited or no experience of using Stata, before creating these documents I was in the same boat! Below, I have listed some great FREE Stata learning resources that I have personally used in my Stata journey. I highly recommend the Stata learning modules from the UCLA Advanced Research Computing – Statistical Methods and Data analytics, as you make use of the Stata pre-set datasets (e.g., auto).

* Stata learning modules via UCLA Advanced Research Computing – Statistical methods and Data analytics: [https://stats.oarc.ucla.edu/Stata/modules/](https://stats.oarc.ucla.edu/stata/modules/)
* Stata training webinars: [https://www.Stata.com/training/webinar/](https://www.stata.com/training/webinar/)

*Note: These training webinars are in CST time so be sure to convert the time to your local time zone.*

* Stata downloadable cheat sheets, these are great if you are a visual learner: [https://www.Stata.com/bookstore/Stata-cheat-sheets/](https://www.stata.com/bookstore/stata-cheat-sheets/)
* You-tube videos. A great example is the 1hr 43min Data for Development YouTube video found: [STATA for beginners course: Stats basics, creating variables, data entry, descriptive stats - YouTube](https://www.youtube.com/watch?v=gdnDkjoPJTM)

## How to use the training materials

In each chapter, there will be descriptions, example code in boxes and do-file objects.

The Stata example code is included in text boxes like:

example code

The text in these boxes are also colour coded:

* Blue – Stata commands, usually at the beginning of a line of code.
* Green – Sub commands. These are important commands on the same code line as the blue Stata command.
* Pink – Local macro. Used in loops to refer to a special variable/word.

Local macros can be quite confusing, so let’s use a straightforward example. Suppose the following loop is what we want to run:

forvalues i=1/3 {

display `i’

}

The macro is `i’ . This essentially tells Stata that for the values 1,2,3 print the value and re-run the loop till value=3. Therefore, we will get an output that looks like:

1

1

2

3

2

3

Similarly, the do-file objects look like this:

A pixel art of a basket of items

Description automatically generated

Where if you click the object, it will take you to a Stata do-file.

**NOTE: Please save these do-files as a copy onto your relevant secure folder and edit the saved copy.**

# Protocol

When starting to write the protocol, use CPRD guidance and another eRAP (Electronic Research Applications Portal) approved form as an example. It’s easier than starting from scratch and some of the titles for the boxes don’t really give you a good idea of what is needed to get the protocol approved. Using an approved example indicates the level of detail needed for that section.

Depending on how complex the study is and whether you are working from a detailed plan in a grant application or similar, it might be easiest to write the methods out in your own preferred style first and then ‘translate’ it to the eRAP form.

The eRAP contains valuable information that we can directly use. I suggest listing the variables you need as it converts a very long written form to a concise variable list.

Example: CPRD Guidance

The CPRD guidance shows generally what type of information is needed in each section. The link to this guidance can be found: <https://cprd.com/guidance-completion-cprd-research-data-governance-rdg-application>, or alternatively under ‘Related resources’ after signing into the CPRD application portal.

There is also general information for these research applications found <https://cprd.com/research-applications>.

CPRD also include a summary of all approved protocols which can be found <https://cprd.com/approved-studies-using-cprd-data>. This is great for checking if there are similar studies to your study.

Tips:

* It can be helpful going in the order of: Background, Study Population then listing of variables.
* Example code lists should be included in the protocol, whether it’s directly included or referenced from a repositry. (These don’t have to be finalised codelists, as long as they show generally which codes you will be using)
* Use an eRAP example like the study type in your project. It can help indicate the level of detail needed for each section. Ask around your institution for an example, others that have worked with CPRD before may allow you to view their submitted eRAP.
* Using a tracking document can be helpful in keeping track of everything. (See below)

### Tracking document:

An excel document is provided below that is a good tool to use whilst doing your CPRD Aurum project. Making note of codelists/where they are stored/covariates etc can help track progress but is also beneficial at the reviewing stage of your completed project.

You may have reviewer comments asking specific questions on code lists, therefore tracking where codelists are stored and timelines are helpful. It is always beneficial to make a note of what you have done and where it’s stored, as these things are usually overlooked/forgotten when nearing the end of a project.



# Code lists

* + What are code lists?
  + Where do I find code lists?
  + How to produce code lists

We will be using the CPRD Aurum code browser file that includes the code browser and the Product and Medical txt files. You can save the browser to your secure Microsoft Team’s folder; it is recommended to have the browser version aligned with the Aurum data cut you use, similar with .txt files too. E.g., if you use May 2022 Aurum data, you should use May 2022 browser or .txt files for generating a codelist. Contact the Data Manager or CPRD key fob holder listed on the eRAP for browser versions.

Reading the CPRD Aurum Specification Document is recommended.

## What are code lists?

The CPRD resource module ‘Defining your study population’ has a thorough explanation of what code lists are. This can be found at: <https://cprd.com/defining-your-study-population>.

Essentially, we want to create files with the terms we are looking for (and their respective codes). For example, if you are asked to create a codelist for Gabapentinoid (class of drugs). Once we have searched for the related terms in the browser or Product txt file, we will end up with a file with all Gabapentinoid related codes and descriptions.

Code lists need to be reviewed with UK clinicians until a finalised version is formed.

The definition of the population needs to be carefully and clearly defined in the eRAP and you may need to have a discussion with the Data Manager or CPRD key fob holder to ensure it is correctly specified.

When the codelists for defining your population are complete, send these to the Data Manager or CPRD key fob holder mentioned in your eRAP. These codes can be sent as a link to a folder in the secure Teams or attaching the code list excel files to an email. After this, you will receive your data from the Data Manager or CPRD key fob holder.

## Where do I find code lists?

There may be pre-existing code lists for the terms you are after, it is beneficial to utilise the Keele EHR (Electronic Health Record) Group Teams and the internet as it saves time.

Note: Code lists may need to be updated depending on the browser and .txt file version they have used to generate the codes. Using a suitable browser and .txt file version when searching for codelists is beneficial, as these may not need to be updated. Section 3.3 covers how to create a codelist, to update a codelist you can use a similar approach.

OpenCodelists is a good website to use when searching for pre-existing codelists, also using university repositories can be helpful.

For ICD-10/ ICD-11 codes, [https://ICD.who.int/en](https://icd.who.int/en) is a website from the World Health Organization that has ICD-10 and ICD-11 browser and Coding Tool. There is also Training available that is beneficial for understanding the structure of ICD-10 / ICD-11 codes as they take the form of ‘mother’ and ‘child’.

The NHS TRUD website has been recommended for finding ICD-10 codes as you can search the terms you need using .txt files instead of using an online browser. Searching using .txt files means your codes can be reproduced, information on how to do this is covered in Section 3.3 under ‘.txt file search’. Creating an NHS TRUD account and subscribing to the latest NHS ICD-10 Edition data files will allow you to access this.

An example of finding codelists using the NHS ICD-10 Edition data files is included in Chapter 8.

For mapping SNOMED CT to ICD-10-CM, I-Magic (Interactive Map-assisted Generation of ICD codes) can be used.

## How do I produce code lists?

This section will cover how to produce code lists using .txt file search and browser search. .txt file searching is recommended as you can re-produce codelists using do-files/scripts which is useful when working on another CPRD project. I personally prefer .txt file searching as the Stata code can be easily updated and reviewed.

In the CPRD Aurum code browser file there are 3 important files:

* Code Browser.
* CPRDAurum Medical (.txt file) – This includes medical history observations.
* CPRDAurum Product (.txt file) – This includes drug and appliance prescription codes.

**Important advice from CPRD:**

1. *Use the version of the CPRD Aurum code browser which corresponds to the CPRD Aurum release you are using for your study;*
2. *Use reusable search strategies for code list generation and maintenance, and;*
3. *When importing codes into the code browser on the online portal, that you double check that all codes have been imported successfully*

Browser Guide:

There is a ‘Code Browser User Guide’ (or alternatively named Research Application User Guide) available in the CPRD CodeBrowser file. This document shows a walkthrough of the interface and visuals on how to use the Browser. If this document is not available in the CPRD CodeBrowser file, contact the Data Manager or CPRD key fob holder asking for the User Guide or download from the CPRD ‘defining your study population’ webpage.

It is beneficial to have a version of the CPRD Aurum Browser aligned with the Aurum data cut you use.

**Important note: Make sure you unlock case sensitive in the browser. Instructions on how to do this is found in section 4.5 in the Code Browser User Guide.**

.txt file search:

The term ‘.txt file search’ is used for searching through the Medical and/or Product .txt files using software (e.g., Stata, R, Python…). This differs from using the Browser as you are essentially writing a do-file/script to replicate what the Browser is doing. It is advised to check terms using the Browser in case there are related terms not included in your codelist. E.g. Searching for prodcode associated with ‘osteoarthritis’, using N05 term to pick up alternative labels such as osteoarthrosis.

Below are examples of creating .txt file code lists step-by-step with Stata code attached.

**IMPORTANT: It is essential to thoroughly check your codelist with clinicians including at least 1 GP in your study team. This is to ensure that the codes included in the codelist are fully representative and accurate to the term searched. I like to send my codelist back-and-forth till a complete finalised codelist is created.**

### **Example: Creating a Gabapentinoid code list.**

Before undertaking any searching, I always note down ‘like’ terms. As I do not have clinical knowledge, it is beneficial to consult with a clinician for terms that need to be included or excluded.

In this case, the Gabapentinoid definition is split into gabapentin and pregabalin as stated in the (hypothetical) eRAP. This eRAP also states that we only want **oral gabapentinoids**.

Searching ‘Gabapentin’ in Google and the NHS website shows that it is also sold under a brand name ‘Neurontin’. This means that we need to add the term ‘Neurontin’ to our search strategy. Doing the same with Pregabalin, we add ‘Lyrica, Alzain, Axalid, Lecaent and Rewisca’.

Below are images of where to find the Brand name on the NHS website, and Brand names from Search engine Bing.

A picture containing text, screenshot, font

Description automatically generated

*Picture 1: Screenshot of NHS website for Gabapentin, found:* [*Gabapentin: medicine to treat epilepsy and nerve pain - NHS (www.nhs.uk)*](http://Gabapentin:%20medicine%20to%20treat%20epilepsy%20and%20nerve%20pain%20-%20NHS%20(www.nhs.uk))

A screenshot of a medical information

Description automatically generated with low confidence

*Picture 2: Screenshot of Bing search for Pregabalin, found:* [*Pregabalin - Search (bing.com)*](https://www.bing.com/search?q=Pregabalin&qs=n&form=QBRE&sp=-1&ghc=1&lq=0&pq=pregabalin&sc=7-10&sk=&cvid=899ED96FC64E4870921BD40EDD23A6D2&ghsh=0&ghacc=0&ghpl=)

Now we have a list of terms to search for, this includes:

* **Gabapentin**
* **Neurontin**
* **Pregabalin**
* **Lyrica**
* **Alzain**
* **Axalid**
* **Lecaent**
* **Rewisca**

After consulting with a GP on the study team, they have approved these terms for inclusion terms.

We are now ready to search using .txt files!

**Step 1: Set your working directory and log.**

Setting a working directory in Stata can make importing files a lot simpler. We also want to log our codelist so that every step is recorded and can be looked at.

Setting a working directory:

cd “My file path:\...\teams folder name\CPRD\_CodeBrowser\_20211\_Aurum\CPRD\_CodeBrowser\_202211\_Aurum”

Using log:

log using “My file path:\...\teams folder name\CPRD\_CodeBrowser\_202211\_Aurum\CPRD\_CodeBrowser\_202211\_Aurum\Logs\nameoflog”

For this example, we are using code browser version 202211.

**Step 2: Importing the Product file**

This step can also be used to import the Medical file but with the file name changed to “CPRDAurumMedical.txt” in the Stata code. We **must import** the long numbers as **string** so that Stata is able to read them, as habit I always import all variables as string.

Note: Import and export in **string,** therefore the long numbers remain intact in the excel document. This is easily done in Stata by importing all string and not changing the format of any variables.

import delimited using “CPRDAurumProduct.txt”, stringcols(\_all) bindquote(nobind)

De-constructing this code:

delimited – This is used for importing .txt files in Stata.

stringcols(\_all) – This is used to import everything in **string** format.

bindquote(nobind) – This allows Stata to ignore trying to match quotation marks to create a full quotation. This is used as some term texts include “ as a measurement without it being a quote. However, this may not be necessary and can potentially mess up the import. It is important to check each file has opened properly.

**Step 3: Setting terms to lowercase**

Since searching is case sensitive, we want to search freely without worrying about including capital letters. We do this by creating new variables.

gen emisterm\_lower = ustrlower(termfromemis)

gen productname\_lower = ustrlower(productname)

gen drugname\_lower = ustrlower(drugsubstancename)

We have created 3 new variables that we can now search in. We want to search in all 3 of these terms just in case the term is matched in one and not the others (Missing terms). If we are specifying a certain type of formulation, we may also make a new variable using ustrlower(formulation).

**Step 4: Searching for the terms**

To refresh, we created a list of terms to search for including Gabapentin, Neurontin, Pregabalin, Lyrica, Alzain, Axalid, Lecaent and Rewisca. Let us create a new variable named poss\_gabapentinoid that represents 1 if we think it is a possible gabapentinoid code and missing otherwise.

gen poss\_gabapentinoid = .

For each of the terms we are searching for, we add asterix’s on either side to let Stata know that we want anything that has gabapentin in, regardless of terms on the other side. E.g., if a term was named ‘100mg gabapentin tablet’ then searching “\*gabapentin\*” will result in a match, however searching “gabapentin” will not.

Term = Gabapentin:

replace poss\_gabapentinoid = 1 if strmatch(emisterm\_lower, “\*gabapentin\*”)

replace poss\_gabapentinoid = 1 if strmatch(productname\_lower, “\*gabapentin\*”)

replace poss\_gabapentinoid = 1 if strmatch(drugname\_lower, “\*gabapentin\*”)

Now we have a 1 in poss\_gabapentinoid if the terms in the 3 specified columns have the word gabapentinoid in.

We do the same for the rest of the search terms. You can repeat the code above for all the inclusion terms or it may be easier to use a loop when searching for many terms.

foreach i in “\*neurontin\*” “\*pregabalin\*” “\*lyrica\*” “\*alzain\*” “\*axalid\*” “\*lecaent\*” “\*rewisca\*” {

replace poss\_gabapentinoid = 1 if strmatch(emisterm\_lower, “`i’”)==1 | strmatch(productname\_lower, “`i’”)==1 | strmatch(drugname\_lower, “`i’”)==1

}

The loop can be used for all search terms by adding “\*gabapentin\*” to the loop.

**Step 5: Keep wanted codes and export**

We have a variable named poss\_gabapentinoid that is 1 if there is a match and missing if not. We want to keep those that are a match. However, we only want oral gabapentinoids.

keep if poss\_gabapentinoid == 1

tab formulation\_lower

drop if formulation\_lower == “gel”

drop emisterm\_lower productname\_lower drugname\_lower poss\_gabapentinoid formulation\_lower

Using the ‘tab’ command lists all formulations used. In this case, we had gel and oral.

Now we only have those that are a match and have dropped the variables created so that the codelist is clean. Let us export!

export excel using “My file path: . . . *file path that you want to export to in the teams folder*”, firstrow(variables)

As mentioned in Chapter 1, I keep these exports saved in the Codes folder.

Now that the codelist is exported, we can close the log.

log close

**Don’t forget to record everything in the Do-File and save this in the team’s channel. I like to save them in the Codes folder as a separate ‘do-file’ folder.**

The exported excel file is the code list! Save your codelist as a Stata file too therefore we can use it to extract in the next chapter.

Below is an icon including the above code in a Stata do-file. Double click to open and **save a copy**. **[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



Note: What if we wanted to widen out our search to find all terms with the same concept id?

Suppose we want to search for terms that have the same snomed concept id as the term gabapentin. We can do this by adding a few lines of code:

gen potential=strmatch(term\_lower, "\*gabapentin\*")

bysort snomedctconceptid: egen check=max(potential)

browse term\_lower if check==1 & pot==0

Here, we create a new variable named ‘potential’ that has all terms that match \*gabapentin\* (similar to what we did above).

The next line is slightly more complicated – we are searching by snomedconceptid and generating a new variable ‘check’ which is the max of the variable potential. For help understanding bysort, search help bysort into Stata.

We then browse the term if there is a match to the snomedconceptid but no match for the term \*gabapentin\*. I.e. the additional terms we may want to add to our codelist.

# Extracting observations and prescriptions using code lists

Let us say we have all the codelists we need in Stata files. Now, we must extract these codes from the CPRD Aurum file folders needed.

E.g., If we want to extract the Gabapentin codelist, we use the Drug Issue file folder and similarly if we want to extract (for example) an Osteoporosis codelist, we use the Observation files folder.

As we are using a lot of loops and different file paths, we can set up ‘Globals’ to make it a lot easier.

A global macro is something that is stored in Stata memory which can be used anytime by reference to its name [1].

For this example, I have stored the Stata files in an extra file named ‘data’ within a Codelists file within the CPRD data file.

global cons “My file path:\....\CPRD data\Observation\”

global codes “My file path:\....\CPRD data\Codelists\data\”

global drugs “My file path:\....\CPRD data\Drug Issue\”

### **Example 1: Extracting the Gabapentinoid codelist.**

**Step 1: Format the Drug Issue .txt files.**

The files in the CPRD data folder are very large, so they are split up into multiple .txt files. The number of .txt files differ depending on your defined study population. For this example, we have 13 Drug Issue .txt files that need to be converted to Stata files so Stata can understand them.

We use two loops as the file names slightly differ, note that single digit extracts are named “\_001” whilst double digits are named “\_010”. As the number of zeroes are different, we take this into account and create a single master name, in this case I used DI00`n’ where n = 1…13.

For this example, we only want those that have received a gabapentinoid prescription.

forvalues i=1/9 {

import delimited using “$drugs\IS\_IA\_ ...\_Extract\_DrugIssue\_00`i’.txt, stringcolls(\_all) bindquote(nobind)

save “$drugs\DI00`i’.dta”, replace

clear

}

forvalues i=10/13 {

import delimited using “$drugs\IS\_IA\_ ...\_Extract\_DrugIssue\_0`i’.txt, stringcolls(\_all) bindquote(nobind)

save “$drugs\DI00`i’.dta”, replace

clear

}

After running this code, we will have 13 Drug Issue Stata files named DI001 up to DI0013.

**Step 2: Extract the gabapentinoid codelist.**

We want to extract the codelist from each DI00 file, so that we only have those that have a gabapentinoid code.

forvalues i=1/13 {

use “$drugs\DI00`i’.dta”, replace

merge m:1 prodcodeid using “$codes\gabapentinoids.dta”, keep(3)

drop \_merge

save “$drugs\gabapentinoid00`i’.dta”, replace

}

clear

Here, we load the first Drug Issue Stata file and merge with the gabapentinoid codelist by procodeid. Since we only want those that have a gabapentinoid prescription we keep the matched merge.

Dropping the \_merge variable keep’s the data clean and then we save it as a new Stata file. This is done for all 13 Drug Issue files. This can equally be done by adding ‘nogen’ to the end of the merge line instead of the drop \_merge in the line below.

We now have 13 Stata files that have the matched drug issue and codelist data. Let us append these to create one useable dataset.

use “$drugs\gabapentinoid001.dta”, clear

forvalues i=2/13 {

append using “$drugs\gabapentinoid00`i’.dta”

}

This creates one dataset with all 13 datasets stacked together in one file.

**Step 3: Converting string dates to Stata readable dates.**

At the moment, all our data is in string format as this allows long integer values to be imported.

We need to convert the dates so that we can use them for later analysis.

gen issuedate1 = date(issuedate, “DMY”)

format issuedate1 %td

gen enterdate1 = date(enterdate, “DMY”)

format enterdate1 %td

Alternatively, you can use the ‘replace’ command to overwrite the current issuedate with the formatted issuedate (and similarly for enterdate).

We may also have issuedate that is missing, this may become an issue later as we use the issuedate variable. To reduce the missing in the issuedate variable, we replace with enterdate if it is missing.

replace issuedate1 = enterdate1 if issuedate1==.

**Step 4: Checking for duplicates.**

We now check for any duplicated rows of data:

duplicates report patid issueid issuedate1

If a table with only ‘1’ appears, then there are no duplicates. If there are duplicates, it is beneficial to have a look at the duplicates and discuss with clinicians, using their judgement whether they are duplicated by the computer or whether multiple identical prescriptions were issued.

**Step 5: Save the file to Working folder.**

I have found it easier to have a folder dedicated to cleaned, ready to use CPRD Aurum dataset’s and to create a global to the ‘working’ folder.

save “My file path:\....\CPRD data\Stata files\Working\gabapentinoid.dta”

### **Example 2: Extracting an Osteoporosis codelist.**

This example is similar to Example 1 but uses the Observation file to extract from. We are assuming the osteoporosis codelist exists as a dataset, if you are unsure on how to create a codelist refer to Chapter 3.

**Step 1: Format the Observation .txt files.**

In this example, we have 16 Observation extract .txt files. We need to convert these to Stata files so that we can use them to extract the osteoporosis codelist. As mentioned previously, we use two loops due to the nature of the zeroes in the names of the Observation files.

forvalues i=1/9 {

import delimited using “$cons\IS\_IA\_ ...\_Extract\_Observation\_00`i’.txt, stringcolls(\_all) bindquote(nobind)

save “$cons\O00`i’.dta”, replace

clear

}

forvalues i=10/16 {

import delimited using “$cons\IS\_IA\_ ...\_Extract\_Observation\_0`i’.txt, stringcolls(\_all) bindquote(nobind)

save “$cons\O00`i’.dta”, replace

clear

}

**Step 2: Extract the Osteoporosis codelist.**

We want to extract the osteoporosis codes from the Observation data, so we are only left with Osteoporosis Observations.

forvalues i=1/16 {

use “$cons\O00`i’.dta”, replace

merge m:1 medcodeid using “$codes\osteoporosis.dta”, keep(3)

drop \_merge

save “$cons\osteoporosis00`i’.dta”, replace

}

clear

Then appending to create one dataset:

use “$cons\osteoporosis001.dta”, clear

forvalues i=2/16 {

append using “$cons\osteoporosis00`i’.dta”

}

This creates a dataset with all 16 matched data together.

**Step 3: Converting dates to Stata readable dates.**

At the moment, all our data is in string format as this allows long integer values to be imported.

We need to convert the dates so that we can use them for later analysis.

gen obsdate1 = date(obsdate, “DMY”)

format obsdate1 %td

gen enterdate1 = date(enterdate, “DMY”)

format enterdate1 %td

We may also have obsdate that is missing, this may become an issue later as we use the obsdate variable. To reduce the missing in the obsdate variable, we replace with enterdate if it is missing.

replace obsdate1 = enterdate1 if obsdate1==.

**Step 4: Duplicates report.**

Sometimes, when appending multiple files, it can generate a duplicate row. We can check these with a line of code:

duplicates report patid obsid obsdate1

If a table with only ‘1’ appears, then there are no duplicates. If there are duplicates, it is beneficial to have a look at the duplicates and take necessary action.

**Step 5: Saving the file to Working folder.**

save “My file path:\....\CPRD data\Stata files\Working\osteoporosis.dta”

Below is an icon including both examples above in a Stata do-file. Double click to open and save a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



# Study population

At this stage we have extracted all the potentially relevant rows for our study from the Observation and Drug Issue files as Stata files and the next step is to define our population. This section relates to the ‘Definition of the study population’ section in the eRAP.

We are going to format the Patient and practice file so that it is useable to use for later.

## Patient file

The Patient file contains basic patient information (demographics and registration details). We need to format this file so that we can easily use it when merging into our study population definition.

**Step 1: Import the Patient .txt file:**

import delimited “My file path:\...\Patient\_001.txt”, stringcols(1)

Replace with your file path and change the name of the Patient\_001 .txt file to the one matching your patient file. stringcols(1) or stringcols(\_all) can be used however\_all is recommended. When I imported this file into Stata only the first column was non-string, therefore using stringcols(1) changes the first column to string.

**Step 2: Dropping unacceptable records:**

Keeping uncertain gender records and erroneous records affects data quality, so we must drop these records.

Gender:

drop if gender!=1 & gender!=2

Erroneous records:

drop if acceptable!=1

Our Patient data now only includes acceptable records and gender noted as 1 or 2. This is standard for cleaning the Patient .txt file.

**Step 3: Formatting the dates in Patient file.**

We need the dates to be readable by Stata, therefore we must format the variables that include a date, these are: emis\_ddate, regstartdate, regenddate and cprd\_ddate.

foreach date in emis\_ddate regstartdate regenddate cprd\_ddate {

gen `date’1 = date(`date’, “DMY”)

format `date’1 %td

drop `date’

rename `date’1 `date’

}

This loop changes emid\_ddate, regstartdate, regenddate and cprd\_ddate to Stata readable dates, converting them from string to dates. This file is now formatted and cleaned, therefore can be saved in the Working folder.

**Step 4: Save in Working folder.**

We want to save this ‘cleaned’ dataset to our Working folder so that it is easier to access and reference throughout the process.

save “My file path:\...\Stata files\Working\patients.dta”

## Practice file

The practice file contains practice information including practice region, identifier, and last collection date. Last collection date is important as it will be used to create an end date for your study population (we will cover this later).

**Step 1: Import Practice .txt file**

This is found in the Practice folder, please replace the file path with your file path.

import delimited “My file path:\...\Practice\_001.txt”, clear

**Step 2: Format dates**

We want to create Stata readable dates, lcd refers to the last collection date.

gen lcd1 = date(lcd, “DMY”)

format lcd1 %td

drop lcd

rename lcd1 lcd

drop uts

Up-to-standard is not currently populated in CPRD Aurum, it is populated in GOLD. Therefore, we drop this variable.

**Step 3: Save in Working folder.**

save “My file path:\...\Stata files\Working\practice.dta”

Below is the Stata do-file for the code above. Save as a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



* 1. Start and End variables:

Suppose in the eRAP, the start date of our study is 01/01/2004 and the end date is 07/10/2022. We need to create ‘End’ and ‘Start’ variables.

* ‘Start’ variable is the latest of the dates: registration start date, pseudo 18th birthdate and start of the study.
* ‘End’ variable is the earliest of the dates: registration end date, date of death, CPRD date of death and the end of the study.

Note: date of death and CPRD date of death are separate variables. For more information, visit the CPRD Aurum Data Specification.

Setting up globals:

global working " My file path:\...\CPRD data\Stata files\Working\"

**Step 1: Import patient file.**

This is the formatted patient file in Chapter 5.1.

use "$working\patients.dta"

**Step 2: Merge with practice file.**

This is the practice file in Chapter 5.2.

merge m:1 using "$working\practice.dta"

Drop \_merge==2. These are those practice records that are not matched to the patient file.

drop if \_merge==2

drop \_merge

**Step 3: Creating start and end variables.**

First, we want to create a pseudo 18th birth date. In this example, we choose this date to be 1st July of the year of birth due to it being in the middle of the year.

Note: Alternatively, you can use 1st January of the year. It is best to choose a date that is consistent with the date used in your team and the wider team.

gen date18 = mdy(7,1,yob)

Creating the start date, reminder that this is the latest of registration start date, pseudo 18th birthday and start of the study. Note: the start of the study is 01/01/2004 as stated at the beginning of Chapter 5.3.

gen start = max(regstartdate, date18, date(“1/1/2004”, “DMY”))

Creating the end date, reminder this is the earliest of date of death, registration end date, CPRD death date and the end of the study.

gen end = max(emis\_ddate, regenddate, cprd\_ddate, date(“7/10/2004”, “DMY”))

**Step 4: Save in working folder.**

save “working\patient\_and\_practice.dta”

Below is the Stata do-file for the code above. Save as a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



# Units

Look-up files include extra information that needs merging into your data if needed. They consist of additional .txt files that may need to be requested from your Data Manager or CPRD key fob holder. For more information, read the CPRD Aurum Data Specification, particularly at the ‘Mapping’ column. In the below example, we will be merging the look-up file numunits to explore the unit terms for eGFR.

Most units may need conversion to metric units. E.g., inches to centimetres, mmol/mol to %, feet to meters etc.

Below I go through an example using eGFR (Estimated Glomerular Filtration Rate) with restriction to CKD (chronic kidney disease) stages. eGFR is measured in ml/min/1.73m² as standard.

### **Example: eGFR Units with restriction**

Let us say we have created an eGFR codelist and have our Working file’s ready to use. After looking at our eRAP, it states we only want those with Stage 3 CKD or above. We use the definition below:

**Definition:** Stage 3 CKD or above defined as eGFR <60.

**Units:** ml/min/1.73m²

Therefore, we want to restrict our eGFR Working data to values <60 (or equivalent in different units).

Setting up LookUp global:

global lookup “My file path:\...\CPRD data\Lookup files\”

**Step 1: Importing NumUnit**

import delimited “$lookup\NumUnit.txt”, stringcols(\_all)

save “My file path:\...\Lookup files\numunit.dta”

Importing and saving the NumUnit as a Stata file so that we can use it to merge into our eGFR data.

**Step 2: Merging NumUnit with eGFR data**

Loading the eGFR data and merging the NumUnit Stata dta file created above.

use “$working\egfr.dta”, clear

merge m:1 numunitid using “$lookup\numunit.dta”

**Step 3: Exploring the units.**

In this step we explore the different units used in this dataset. Firstly, we replace values that are NA to be missing.

replace value=“” if value==“NA”

Changing format of value from string to numeric using destring Stata command.

destring value, replace

Looking at the types of units that are included in the dataset and dropping if missing and 0 as these do not meet our definition.

tab description

drop if value==.

drop if value==0

Suppose that when we run ‘tab description’, we find the units are:

* s
* ratio
* see \*
* .
* See below
* UNKNOWN UNITS
* Units
* ml/min/1.73m2
* ml/min/1.73m^2
* ml/min/1.73m 2

From our unit’s definition, we want to keep the last 4. “Units” is ambiguous but is the mostly used unit, it is worth investigating further.

**Step 4: Keeping suitable units.**

Generate a new variable that is 1 if the unit is unsuitable for our definition, this makes it easier to drop.

gen removeunit = .

replace removeunit=1 if description== “s” | description== “ratio” | description== “see \*” | description== “.” | description== “See below” | description== “UNKNOWN UNITS”

drop if removeunit== 1

The previous step mentioned a unit named “Units” as being the most used. Looking further, we find out that the value range of this unit if 0.1 to 1000. After consulting with the team, we decide to keep this unit but restrict the unit in line with the eGFR definition (i.e. only use values that are within the plausible range for an eGFR measurement in ml/min/1.73m²).

**Step 5: Using the definition.**

We have noted previously that stage 3 CKD or above is <60 eGFR value.

gen possstage3ckd = 1 if value <60

keep if posstage3ckd == 1

save “My file path:\...\Working\egfr\_stage3.dta”

Below is the Stata do-file for the code above. Save as a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



### General metric conversions:

If you need to convert units to metric, I advise using the imbedded unit converter from Google or Bing. Below, I have included some formulas for how the approximate value of each unit is calculated, but it is easier to use the Google/Bing unit converter.

**Weight:** Approx. Kg =

OR, Use Microsoft Bing unit converter. I searched ‘pounds to kg’:

A screenshot of a math test

Description automatically generated

*Picture 3: Screenshot of Bing search for pound to kg converted units.*

**Height**: Approx. Meter =

Approx. Centimeter = inch × 2.54

**Other:** Approx. Litres = Imperial Gallon × 4.546

### **Exercise 1**

This exercise was created to allow you to attempt calculating BMI using weight and height variables. At the end of this exercise, we will be able to calculate BMI from height and weight variables that differ in units. We will be using my version of the numunit .txt look up file below:

Note: When using look-up files for your CPRD Aurum study, use the ones that were provided by the Data Manager or CPRD key fob holder, not the ones included in these training materials.

Please save this as a Stata file as you will need to merge it into Stata datasets.



**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**

Similarly, the excel document needed is below. Note, Sheet 1 refers to Weight and Sheet 2 refers to Height. You will need to load these in separately when completing the below questions.



**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**

To do:

1. Save NumUnit.txt as a Stata file.
2. Import the Weight excel sheet into Stata and merge NumUnit Stata file.
3. Generate a variable named metricweight that includes the weights in units kg (kilograms). You may need to convert some unit’s values to get a kg weight value.
4. What is the mean value of metricweight variable? Also note down the confidence interval of the mean as well as the min and max value. Are there any outlier values?
5. Save this file as you will need to refer to it later.
6. Import the Height excel sheet into Stata and merge NumUnit Stata file.
7. Generate a variable named metricheight that includes the weights in units m (meters). You may need to convert some unit’s values to get a meter height value.
8. Are there any unit’s that may need to be excluded? If so, ensure the metricheight is missing for that unit.
9. What is the mean value of metric height? Also note down the confidence interval of the mean as well as the min and max value. Are there any outlier values?
10. Merge in the Weight Stata file you saved in Step 5.
11. Generate a BMI variable where BMI =
12. What is the mean BMI value?

Solutions and do-file to the questions are in Chapter 9.

# Defining other variables

In this chapter we will go over how to create general covariates, e.g., Age, Gender, Region. There is an exercise at the end of this chapter that allows you to generate these covariates in a dataset I created. We will be using my version of the Gender and Region lookup file which is included in the training materials. You can import these into Stata to have a look at what the dataset looks like.

Note: When using look-up files for your CPRD Aurum study, use the ones that were provided by the Data Manager or CPRD key fob holder, not the ones included in these training materials.

Age at index date:

We have previously created a pseudo birthdate that refers to the 1st July of the birth year. Refer to Chapter 5.3 Step 3.

Suppose we want to create a new variable which is the age at the index date (Let’s suppose the index date is the age at which a patient had a certain prescription for the first time).

Indexdate refers to the date of certain first prescription and date18 as the pseudo birthdate.

gen indexage = (indexdate – date18)/365.25

summ indexage

We now have an age variable. What if we wanted the truncated age?

gen trunc\_age = trunc(indexage)

Gender:

This is a .txt look-up file that includes categories of gender in CPRD Aurum. As Stata does not merge in .txt files, you will need to save this as a Stata data file (.dta).



**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**

To save the .txt file as a Stata file, use the below code:

Note: We do not need to import all variables as strings.

import delimited " My file path:\...\Gender.txt"

save “My file path:\...\Working\Gender.dta”

OR

A screenshot of a computer

Description automatically generatedUsing the Stata interface:

*Picture 4: Screenshot of Stata interface in dark mode with arrow pointing to Text data.*

Select File → Import → Text data (delimited, \*.csv, …)

Then select Save from the File tab.

Region:

This is a .txt look-up file that includes categories of Region in CPRD Aurum. Save this .txt file as a Stata data file (.dta) as Stata does not merge in .txt files.



**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**

### **Exercise 2**

In this Exercise you are going to merge in lookup files provided above and create an age variable. There is solution code in a do-file and solutions to the questions in Chapter 9. Load your saved version of the excel document below. Stata will not load the original version as it is protected – make sure you save a copy to your drive.



**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**

To do:

1. Create an age variable assuming this year is 2023. Find the mean value of the age (along with the confidence intervals) and the minimum and maximum values. Are there any outlier age values? If so, which patid(s) have outlier age values?
2. Merge in the Gender lookup file. What % of people are female in this dataset? Are they any that are not Male or Female?
3. Merge in Region lookup file. How many are from the London region? How many are from the West Midlands region?
4. Save your finalised dataset and compare your results to the solutions in Chapter 9.

# HES APC

This chapter will walkthrough the steps to generate a codelist to use for HES APC (Hospital Episode Statistics Admitted Patient Care) and how to extract them. We will be using ‘fracture’ to generate ICD-10 and OPCS codes and extracting these from the Diagnoses and Procedures file in HES APC.

ICD-10 codes differ from those found in Aurum, as these are used in secondary care linked data. To start, you must register a TRUD account at NHS Digital. After you have registered an account, you want to subscribe to ‘NHS ICD-10 5th Edition data files’ and ‘OPCS-4 data files’. This will allow you to download the data files needed for .txt file code searching, similar to the CPRD Aurum code lists.

ICD-10 is used for diseases and other health conditions whilst OPCS-4 is used to classify surgical procedures and interventions (in secondary care) [2].

If you would prefer to use a browser, you can use the ICD-10/11 (more updated) browser found [https://ICD.who.int/browse11/l-m/en](https://icd.who.int/browse11/l-m/en).

**Note: When using linked data, make sure you note the coverage period as this may affect your end date.**

## ICD-10 Codelist

To understand the relationship and structure of the ICD-10/11 codes, go to the WHO (World Health Organisation) training materials page. This includes walk-through videos and interactivity in all the Units. Link: [https://ICDcdn.who.int/ICD11training/index.html](https://icdcdn.who.int/icd11training/index.html).

There is also a training unit on how to use the ICD-11/10 browser following the same link as above.

Before embarking on code searching, it is always beneficial to have an exclusion terms list. For the example below, suppose the study team decided that we want ICD-10 fracture codes but we want to exclude \*tooth\* and \*birth\* terms. Hence, we don’t want to include tooth fractures or birth fractures in our codelist.

Note: When searching for ICD-10 codelists, the same term used for generating CPRD Aurum Medical/Product codelists may not translate across, therefore it is beneficial to browse on description to check if there is a match. E.g., If creating a fracture codelist in Medical Aurum file, we can add the extra exclusion of \*dental\* in relation to \*tooth\*. However, in the ICD-10 data file, there are no matches for \*dental\*, hence it is not included in the exclusion terms for the fracture codelist.

### Synonym searching:

When searching for codes, it is advised to search for like-terms. An example of this, is using the ICD-10 WHO browser with ‘synonym’ box ticked. This ensures that the term searched, and like-terms are shown. Below I have included an example of the term fracture and kidney. Fracture does not have any like-terms in the ICD-10 who browser database, so I tried searching for ‘break’ and ‘crack’, but these did not return any fracture related codes. Kidney has a few synonyms, one being renal which is shown in the images below. The ICD-10 browser link is below if you would like to try yourself.

[ICD-10 Version:2019 (who.int)](https://icd.who.int/browse10/2019/en#!)

Let’s search for fracture. Make sure you click ‘Advanced search’ as this allows you to search for synonym terms. I have included a picture of the advanced search box I used with the relevant boxed ticked.

A screenshot of a computer

Description automatically generated

*Picture 5: Screenshot of ICD-10 browser with search for fracture term.*

Pressing ‘Search’ will yield a lot of terms with fracture highlighted in bold. However, there are no terms other than fracture that is highlighted. Below is a picture of some terms that was shown after searching for fracture and fracture synonym.

A screenshot of a computer

Description automatically generated

*Picture 6: Screenshot of the results window from the ICD-10 browser with search for fracture term.*

So we know that when searching for the term fracture in ICD-10, there are no additional synonym terms that need to be added. You can also click through the ‘tree’ structure of the codes and check term-by-term.

A screenshot of a computer

Description automatically generatedLet’s now look at the term kidney. As before, we make sure the synoym box is ticked.

*Picture 7: Screenshot of ICD-10 browser with search for kidney term.*

A whiteboard with a black frame

Description automatically generatedPressing search yields the results below. Take note of the additional terms in bold: ‘Renal’, ‘Hepatorenal’ and ‘Extrarenal’. It is worth investigating these terms further, this can be done by clicking on the term.

*Picture 8: Screenshot of the results window from the ICD-10 browser with search for kidney term.*

A red and yellow rectangular object

Description automatically generatedSuppose we click the second row: Q61.4 **Renal** dysplasia. The following information is displayed:

*Picture 9: Screenshot of the ICD-10 browser results window for term Renal dysplasia.*

This suggests that this code does relate to kidney by looking at the bullet points. It is worth noting the exclusion terms, shown as ‘Excl.:’.

Now let’s look through an example of searching for fracture terms using the NHS ICD-10 data files. Note that we did not find any like-term to fracture, so we will only have \*fracture\* for this example. If we were searching for kidney, we would add \*renal\*, \*extrarenal\* and \*hepatorenal\* terms (after discussing with clinicians).

### **Example: Searching for fracture in the NHS ICD-10 5th Edition data files**

Let us say that we need to generate an ICD-10 codelist for fractures.

Recap:

|  |  |
| --- | --- |
| **Inclusion terms** | **Exclusion terms** |
| \*fracture\* | \*tooth\* |
|  | \*birth\* |

*Table 1: Inclusion and Exclusion terms for fracture ICD-10 codelist*

**Step 1: Import Codes and Titles .txt file.**

We want to use the .zip file downloaded from NHS TRUD. After extracting the contents from this .zip file, we want to go into the ‘Content’ folder and import the ‘CodesAndTitlesAndMetadata’ .txt file.

import delimited “My file path:\...\codes\ICD-10\...\Content\ICD-10\_Edition5\_CodesAndTitlesAndMetadata\_GB.txt”, bindquote(nobind) stringcols(4 6 9 10 11)

**Step 2: Generate lowercase variables.**

We want to generate lowercase variables so that we can freely search without the restriction of case sensitivity.

gen description\_lower = ustrlower(description)

**Step 3: Searching for inclusion terms.**

We generate a new variable named ‘poss\_fracture’ that is 1 if the description matches the search term and missing otherwise.

gen poss\_fracture = .

replace poss\_fracture = 1 if strmatch(description\_lower, “\*fracture\*”)

Keeping only the fracture codes

keep if poss\_fracture == 1

**Step 4: Searching for exclusion terms.**

Generating a new variable named ‘exclude’ so we can easily drop all those in our exclusion criteria.

gen exclude = .

browse if strmatch(description\_lower, “\*tooth\*”)

replace exclude = 1 if strmatch(description\_lower, “\*tooth\*”)

browse if strmatch(description\_lower, “\*birth\*”)

replace exclude = 1 if strmatch(description\_lower, “\*birth\*”)

Dropping the exclusion terms,

drop if exclude == 1

So now you have a Stata file that satisfies the inclusion and exclusion terms! Save this Stata file to your codelist folder. Don’t forget to check the codelist with a GP on the study team.

Below is an icon including the example above in a Stata do-file. Double click to open and save a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



## OPCS Codelist

Let us say we want fracture procedure codes. After discussing with the study team, they have decided on some inclusion and exclusion terms. We are including the terms fracture and vertebr but excluding spine and revisional.

Note how these terms differ from the ICD-10 terms in Chapter 8.1. This is due to OPCS being interventions and procedures codes, therefore a different revised codelist is needed.

The inclusion and exclusion criteria are listed in Table 2:

|  |  |
| --- | --- |
| **Inclusion terms** | **Exclusion terms** |
| \*fracture\* | \*spine\* |
| \*vertebr\* | \*revisional\* |

*Table 2: Inclusion and exclusion terms for fracture OPCS codelist*

### **Example: Searching for fracture in the NHS OPCS 4 data files**

**Step 1: Import Codes and Titles .txt file.**

Similar to the ICD-10 codelist, we want to use the .zip OPCS file downloaded from NHS TRUD. After extracting the contents of the .zip file, we want to use the Codes and Titles file. In this example we use version 4.10 (Nov 2022).

import delimited "My file path:\...\codes\ICD-10\nhs\_opcs4df\_10.0.0\_20221101000001\OPCS410 Data files txt\OPCS410 CodesAndTitles Nov 2022 V1.0.txt", bindquote(nobind) stringcols(\_all)

**Step 2: Generate lowercase variables.**

Generating lowercase variables allows us to search free from case sensitivity restrictions.

gen v2\_lower = ustrlower(v2)

**Step 3: Searching for inclusion terms.**

We generate a new variable named ‘poss\_fracture’ that is 1 if the description matches the term and missing otherwise. For which terms to search for, refer to the Inclusion terms (Table 2).

gen poss\_fracture = .

replace poss\_fracture = 1 if strmatch(v2\_lower, “\*fracture\*”)

replace poss\_fracture = 1 if strmatch(v2\_lower, “\*vertebr\*”)

Keeping only the inclusion terms

keep if poss\_fracture == 1

**Step 4: Exclusion terms**

We want to create a variable that is 1 if it matches our exclusion terms and missing otherwise. This means that we can use the drop Stata command to create a codelist that conforms to the inclusion and exclusion criteria.

Note: The exclusion terms are \*spine\* and \*revisional\* as noted in Table 2.

gen exclude = .

replace exclude = 1 if strmatch(v2\_lower, “\*spine\*”)

replace exclude = 1 if strmatch(v2\_lower, “\*revisional\*”)

Dropping the exclusion terms

drop if exclude == 1

Now we have a Stata do-file that has fracture OPCS codes. Save this file to your codelist folder and don’t forget to check all codelists with a clinician on your study team.

Below is an icon including the example above in a Stata do-file. Double click to open and save a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



## Patient file HES APC

The patient file in HES APC contains variables related to the matching and ethnicity. It is advised to read the Data Dictionary provided by HES APC in the data download. Below, we will go through how to format the Patient file so that it includes ‘acceptable’ patients.

Setting globals:

global hesapc “My file path:\....\CPRD data\HES APC\Type2\_delivery\Results\Aurum\_linked\Final\”

global working “My file path:\....\Stata files\Working\”

**Step 1: Import HES APC Patient file**

import delimited “$hesapc\hes\_patient.txt”, bindquote(nobind) stripquote(no) stringcols(\_all)

Now that we have imported the data, we want to only keep ‘acceptable’ patients. This includes dropping patients will large HES ID’s (>20), as stated in the data dictionary.

**Step 2: Look at n\_patid\_hes**

codebook n\_patid\_hes

At the moment, all our data is in string format. We want n\_patid\_hes in integer format so we can utilise the drop command without a large list of string values.

gen hesid = real(n\_patid\_hes)

drop n\_patid\_hes

rename hesid n\_patid\_hes

drop if n\_patid\_hes >20

Next, we look at quality of matching between AURUM and HES. It is advised to look at your match\_ranks, in case using the general dropping rank>1 may drop too many patients. Adjust your drop rank criteria to your data.

**Step 3: Look at quality of matching**

codebook match\_rank

Suppose after looking at the values for match rank, we decide to drop those with a drop rank >1 (general criteria).

keep if match\_rank == “1”

If we wanted to drop match rank >2 we can use either:

keep if match\_rank == “1” | match\_rank== “2”

OR

gen matchrank1 = real(match\_rank)

drop match\_rank

rename matchrank1 match\_rank

drop if match\_rank >2

Finally, keep the variables you need for the patient list. In this case, we only want to keep patid as we are going to merge this with our Aurum and HES data and drop unacceptable patients.

keep patid

Saving the list to working folder,

save “$working\HESlinked\_acceptable\_patients\_dta”



## Extracting ICD-10 codelist from HES APC

We will go through examples of extracting a fracture ICD-10 codelist in Primary HES diagnoses, HES diagnoses hospitalisation and HES diagnoses episode.

It is important to refer to the eRAP to which data files you need as some studies may use all the Diagnoses data files from HES APC whilst others only use the Primary HES diagnoses data file.

Note: the data dictionary file includes information on the different data files and data file structure within HES APC, it is advised to read this.

We will be using the ICD-10 command in Stata. To read the help file, run the code below.

help ICD-10

Setting up globals:

global codes “My file path:\....\CPRD data\Codelists\data\”

global hesapc “My file path:\....\CPRD data\HES APC\Type2\_delivery\Results\Aurum\_linked\Final\”

global working “My file path:\....\Stata files\Working\”

### **Example 1: Extracting fractures ICD-10 codelist from Primary HES diagnoses.**

**Step 1: Import hes\_primary\_diag\_hosp.txt**

import delimited "$hesapc\hes\_primary\_diag\_hosp.txt", bindquote(nobind) stripquote(no) stringcols(1 2)

**Step 2: Using ICD-10 command**

Let’s check if the codes in this file are all defined ICD-10 codes

ICD-10 check ICD\_primary, fmtonly summary version(2019)

Changing the ICD-10 codes so that there are no dots. E.g. XXXX format instead of XXX.X

ICD-10 clean ICD\_primary, gen(ICD-10) nodots

We want to rename this variable to match the variable name in the codelist data, so that we can run a merge.

rename ICD-10 alt\_code

**Step 3: Merge with fracture ICD-10 reviewed codelist**

merge m:1 alt\_code using "$codes\fracture\_ICD-10.dta"

**Step 4: Check the merge**

By browsing \_merge==1 and \_merge==2, we can see what terms were not successfully matched.

browse if \_merge==2

browse if \_merge==1

We want to keep those that are matched in both codelist and primary diagnoses

keep if \_merge==3

**Step 5: Tidy the data**

We want to drop the empty variables and merge variable. In this case, the modifier\_4 and modifier\_5 was all missing, therefore it will be dropped.

drop \_merge

codebook modifier\_4

codebook modifier\_5

drop modifier\_4 modifier\_5

Next we want to format the dates, so they are Stata readable. We use a loop to do this:

foreach date in admidate discharged {

gen `date’1 = date(`date’, “DMY”)

format `date’1 %td

drop `date’

rename `date’1 `date’

}

Browse missing dates

browse if admidate==.

browse if discharged==.

**Step 6: Merge with acceptable patients**

This step ensures those in our data have acceptable HES records and acceptable linkage to Aurum (See Chapter 8.3 for more details).

merge m:1 patid using "$working\HESlinked\_acceptable\_patients.dta"

Keep those that are acceptable,

keep if \_merge==3

drop \_merge

Save to working folder,

save "$working\fractures\_ICD-10\_primarydiaghosp.dta"

### **Example 2: Extracting fractures ICD-10 codelist from HES diagnoses hospitalisation.**

**Step 1: Import hes\_diagnoses\_hosp.txt**

import delimited "$hesapc\hes\_diagnosis\_hosp.txt", bindquote(nobind) stripquote(no) stringcols(\_all)

**Step 2: Using ICD-10 command**

Let’s check if the codes in this file are all defined ICD-10 codes

ICD-10 check ICD\_primary, fmtonly summary version(2019)

Changing the ICD-10 codes so that there are no dots. E.g. XXXX format instead of XXX.X

ICD-10 clean ICD, gen(alt\_code) nodots

This code is similar to the previous example; however we skipped the step of renaming the new variable by typing the new variable name into the gen() command.

**Step 3: Merge with fracture ICD-10 reviewed codelist**

merge m:1 alt\_code using "$codes\fracture\_ICD-10.dta"

**Step 4: Check the merge**

By browsing \_merge==1 and \_merge==2, we can see if the merge was successful or not. I do this as a habit to ensure the merge wanted to do as I asked.

browse if \_merge==2

browse if \_merge==1

We want to keep those that are matched in both codelist and diagnoses hospitalisation.

keep if \_merge==3

**Step 5: Tidy the data**

We want to drop the empty variables and the merge variable. In this case, the modifier\_4 and modifier\_5 was all missing, therefore it will be dropped.

drop \_merge

codebook modifier\_4

codebook modifier\_5

drop modifier\_4 modifier\_5

We want to format the dates, so they are Stata readable. We use a loop to do this:

foreach date in admidate discharged {

gen `date’1 = date(`date’, “DMY”)

format `date’1 %td

drop `date’

rename `date’1 `date’

}

Browsing the missing dates

browse if admidate==.

browse if discharged==.

**Step 6: Merge with acceptable patients**

This step ensures those in our data have acceptable HES records and acceptable linkage to Aurum (See Chapter 8.3 for more details).

merge m:1 patid using "$working\HESlinked\_acceptable\_patients.dta"

Keeping those that are acceptable,

keep if \_merge==3

drop \_merge

Saving to the working folder,

save "$working\fractures\_ICD-10\_primarydiaghosp.dta"

Note: this example is similar to the previous example including Primary\_hes\_diagnoses, as they have similar variables.

### **Example 3: Extracting fractures ICD-10 codelist from HES diagnoses episode.**

**Step 1: Import hes\_diagnoses\_epi.txt**

import delimited "$hesapc\hes\_diagnosis\_epi.txt", bindquote(nobind) stripquote(no) stringcols(\_all)

**Step 2: Using ICD-10 command**

Let’s check if the codes in this file are all defined ICD-10 codes

ICD-10 check ICD\_primary, fmtonly summary version(2019)

Changing the ICD-10 codes so that there are no dots. E.g. XXXX format instead of XXX.X

ICD-10 clean ICD, gen(alt\_code) nodots

This code is similar to the first example; however we skipped the step of renaming the new variable by typing the new variable name into the gen() command.

**Step 3: Merge with fracture ICD-10 reviewed codelist**

merge m:1 alt\_code using "$codes\fracture\_ICD-10.dta"

**Step 4: Check the merge**

By browsing \_merge==1 and \_merge==2, we can see if the merge was successful or not.

browse if \_merge==2

browse if \_merge==1

We want to keep those that are matched in both codelist and diagnoses hospitalisation.

keep if \_merge==3

**Step 5: Tidy the data**

We want to drop the empty variables and the merge variable. In this case, the modifier\_4 and modifier\_5 was all missing, therefore it will be dropped.

drop \_merge

codebook modifier\_4

codebook modifier\_5

drop modifier\_4 modifier\_5

We want to format the date variables epistart and epiend, so they are Stata readable. We use a loop to do this:

foreach date in epistart epiend {

gen `date’1 = date(`date’, “DMY”)

format `date’1 %td

drop `date’

rename `date’1 `date’

}

Browsing the missing dates

browse if epistart==. | epiend==.

|

**Step 6: Merge with acceptable patients**

This step ensures those in our data have acceptable HES records and acceptable linkage to Aurum (See Chapter 8.3 for more details).

merge m:1 patid using "$working\HESlinked\_acceptable\_patients.dta"

Keeping those that are acceptable,

keep if \_merge==3

drop \_merge

Saving to the working folder,

save "$working\fractures\_ICD-10\_diagepi.dta"

Note: this example is similar to the two previous examples but has different date variable names.

Below is the Stata do-file for the code above. Save as a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



## Extracting OPCS codelist from HES APC

We want to use the acceptable patients file from Chapter 8.3. Please refer to this chapter if wanting to create an acceptable patient file. Extracting OPCS codes are slightly different from extracting ICD-10 codes.

Setting up globals:

global codes “My file path:\....\CPRD data\Codelists\data\”

global hesapc “My file path:\....\CPRD data\HES APC\Type2\_delivery\Results\Aurum\_linked\Final\”

global working “My file path:\....\Stata files\Working\”

### **Example: Extracting fractures opcs codelist.**

**Step 1: Import and save procedures\_epi.txt**

import delimited "$hesapc\hes\_procedures\_epi\_22\_002482.txt", bindquote(nobind) stripquote(no) stringcols(\_all)

save "$working\HES\_procepi.dta"

clear

We import and save the procedures\_epi so that we can merge in the ‘cleaned’ OPCS fracture codelist.

**Step 2: Run ICD-10 command with nodots option on OPCS codes and tidy**

Import fracture opcs codes

use "$codes\fracture\_opcs.dta"

Replace codes with codes without dots, e.g. XXX.X becomes XXXX

ICD-10 clean v1, gen(opcs) nodots

Rename the variable to match procedures data,

rename v2 description

Save

save "$codes\fracture\_opcs\_cleaned.dta"

**Step 3: Merge codes with procedures data**

We want to merge the ‘cleaned’ fracture codelist and the HES procedures data.

clear

use "$working\HES\_procepi.dta"

merge m:1 opcs using "$codes\fracture\_opcs\_cleaned.dta"

Keep those matched and drop unnecessary variables

keep if \_merge==3

drop \_merge

**Step 4: Format dates**

Running the loop below will format all the dates within the Procedures dataset. Note: There are more date variables than in Diagnoses HES files.

foreach date in admidate epistart epiend discharged evdate {

gen `date’1 = date(`date’, “DMY”)

format `date’1 %td

drop `date’

rename `date’1 `date’

}

Check if there are missing dates,

browse if epistart==. | epiend==. | admidate==. | discharged==. | evdate==.

|

**Step 5: Merge with acceptable patients**

This step ensures those in our data have acceptable HES records and acceptable linkage to Aurum (See Chapter 8.3 for more details).

merge m:1 patid using "$working\HESlinked\_acceptable\_patients.dta"

Keeping those that are acceptable,

keep if \_merge==3

drop \_merge

Saving to the working folder,

save "$working\fractures\_opcs\_procepi.dta"

Below is the Stata do-file for the code above. Save as a copy.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



# Solutions

Exercise 1:

1. **What is the mean value of metricweight variable? Also note down the confidence interval of the mean as well as the min and max value. Are there any outlier values?**

Solution:

Mean weight value: 71.4 (1 d.p)

Confidence interval: [64.5, 78.3]

Min: 55

Max: 93

No, no outlier values.

**8. Are there any unit’s that may need to be excluded? If so, ensure the metricheight is missing for that unit.**

Solution:

Yes, description = “C” for patid = 12.

This is because “C” does not correspond to a height or weight value, instead it may be a temperature value (Celsius).

**9. What is the mean value of metric height? Also note down the confidence interval of the mean as well as the min and max value. Are there any outlier values?**

Solution:

Mean height value: 1.6 (1 d.p)

Confidence interval: [1.6, 1.7]

Min: 1.45

Max: 1.78

No, no outlier values.

**12. What is the mean BMI value?**

Solution:

Mean BMI value: 27.0 (1 d.p)

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



Exercise 2:

1. **Find the mean value of the age (along with the confidence intervals) and the minimum and maximum values.**

Solution:

Mean Age value: 61.7 (1 d.p)

Confidence interval: [20.3, 103.0]

Min: 20

Max: 323

**Are there any strange age values? If so, which patid(s) have the strange age values?**

Solution:

Yes, patid = 8 has age 323.

The oldest person ever recorded is aged 122 years, so this is an outlier [3].

1. **What % of people are female in this dataset? Are they any that are not Male or Female?**

Solution:

53.3% are female. There is 1 patid that is labelled as ‘U’, this is patid = 8.

1. **How many are from the London region? How many are from the West Midlands region?**

Solution:

2 are from London which account for 13.3% of the people in the data.

4 are from West Midlands which account for 26.7% of the people in the data.

**[Please DO NOT EDIT, ONLY EDIT SAVED COPY]**



# Final thoughts

These training materials were created with the intent of aiding your CPRD Aurum journey alongside using CPRD’s own resources. The materials are introductory, and I hope the Stata do-files supplied at each chapter contribute to supporting your Stata coding skills. There are great links in the introduction chapter with Stata coding courses/videos and hopefully in the future these materials may expand to R.

Overtime, these materials will be updated and reviewed as necessary as some links may be broken/incorrect and information may be outdated. If there is any incorrect information currently in the materials or do-files, please don’t hesitate to contact me.

The materials were funded by NIHR SPCR Seed Corn Funding and were created with the help of the Electronic Health Records Group at Keele University.

Abbreviations:

|  |  |
| --- | --- |
| CPRD | Clinical Practice Research Datalink |
| EHR | Electronic Health Records |
| UCLA | University of California, Los Angeles |
| eRAP | Electronic Research Applications Portal |
| GP | General Practitioner |
| NHS | National Health Service |
| eGFR | Estimated Glomerular Filtration Rate |
| CKD | Chronic Kidney Disease |
| BMI | Body Mass Index |
| HES | Hospital Episode Statistics |
| HES APC | Hospital Episode Statistics Admitted Patient Care |

# References

[1] wlm.userweb.mwn.de. (n.d.). *Stata Guide: Macros*. [online] Available at: <https://wlm.userweb.mwn.de/Stata/wstatmac.htm> [Accessed 4 Jul. 2023].

[2] NHS Digital. (n.d.). *Clinical Classifications*. [online] Available at: <https://digital.nhs.uk/services/terminology-and-classifications/clinical-classifications> [Accessed 4 Jul. 2023].

[3] Guinness World Records. (1997). *Oldest person ever*. [online] Available at: <https://www.guinnessworldrecords.com/world-records/oldest-person/> [Accessed 19 Dec. 2023].

‌