Information and Informatics Services, Clinical Research Department, Enterprise Analytics Team

Poojyatha Venkatesh - Data Science Intern, Indiana University Health

Supervisor : Harlan A Nelson

RxNorm is a normalised naming system for drugs. Medical data that was unstructured was downloaded. It was in an XML format that was highly complex. The first task was to flatten these xml tags. What was challenging was that these XML documents did not have the same children tags. Even the children tags did not have a structure that was similar to eachother. Every child tag was different and the same children tags had different structures. For example,

One concept tag would look like this

```
      <concept>
       <namespace>MED-RT</namespace>
       <name>Vascular Endothelial Growth Factor-directed Antibody Interactions
[MoA]</name>
       <code>N0000178291</code>
       <status>A</status>
       <property>
             <namespace>MED-RT</namespace>
             <name>CTY</name>
             <value>MoA</value>
       </property>
       <property>
             <namespace>MED-RT</namespace>
             <name>NUI</name>
             <value>N0000178291</value>
       </property>
       <synonym>
             <namespace>MED-RT</namespace>
             <name>Synonym</name>
             <to_namespace>MED-RT</to_namespace>
             <to_name>VEGF-directed Antibody Interactions</to_name>
             <preferred>false</preferred>
       </synonym>
       <synonym>
             <namespace>MED-RT</namespace>
             <name>Preferred Term</name>
             <to_namespace>MED-RT</to_namespace>
             <to_name>Vascular Endothelial Growth Factor-directed Antibody
Interactions</to_name>
             <preferred>true</preferred>
       </synonym>
      </concept>
```

And another concept tag would look like this :

```
<concept>
       <namespace>MED-RT</namespace>
       <name>Bone Surface Interactions [MoA]</name>
```

```
<code>N0000009958</code>
<status>A</status>
<property>
      <namespace>MED-RT</namespace>
      <name>CTY</name>
      <value>MoA</value>
</property>
<property>
      <namespace>MED-RT</namespace>
      <name>NUI</name>
      <value>N0000009958</value>
</property>
<synonym>
      <namespace>MED-RT</namespace>
      <name>Preferred Term</name>
      <to_namespace>MED-RT</to_namespace>
      <to_name>Bone Surface Interactions</to_name>
      <preferred>true</preferred>
</synonym>
</concept>
```

If you compared the above two code snippets, you will notice that the number of children tags of concept : mainly synonym and property are different in number for the two concept tags.

In such a case, how would you convert these xml tags into flat data structures with rows and columns? We will get to this part in a minute.

Therefore, our task was to first flatten these xml documents. For this we used R Studio.

Load the xml document For loading the xml document, we use a package called xml2. Once we load the xml document, I am going to elaborate on how I flattened the concept tag as the rest of the tags later in a minute but after finding the concept tag and next, we convert this into a tible (something similar to a table), before which we convert the tag into a list via 'as_list'. The beside screenshots explains this process :

```r
d <- xml2::read_xml(here::here("inst/Core_MEDRT_XML/Core_MEDRT_20220502_DTS.xml"))
d %>% head()
```

```
$node
<pointer: 0x0000012ef2979710>

$doc
<pointer: 0x0000012ee871d660>
```

```r
dconcept <-  xml_find_all(d, 'concept')

dconcept %>% head()
```

```
{xml_nodeset (6)}
[1] <concept>\n  <namespace>MED-RT</namespace>\n  <name>1-Compartment [PK]
</name>\n  <code>C298</code>\n  <id>299</id>\n  <status ...
[2] <concept>\n  <namespace>MED-RT</namespace>\n  <name>14-alpha Demethylase
Inhibitors [MoA]</name>\n  <code>C300411</code>\n  < ...
[3] <concept>\n  <namespace>MED-RT</namespace>\n  <name>2-Compartment [PK]
</name>\n  <code>C300</code>\n  <id>301</id>\n  <status ...
[4] <concept>\n  <namespace>MED-RT</namespace>\n  <name>3-Compartment [PK]
</name>\n  <code>C302</code>\n  <id>303</id>\n  <status ...
[5] <concept>\n  <namespace>MED-RT</namespace>\n  <name>4-Hydroxyphenyl-Pyruvate
Dioxygenase Inhibitor [EPC]</name>\n  <code>C635 ...
[6] <concept>\n  <namespace>MED-RT</namespace>\n  <name>5-Lipoxygenase Inhibitor
[EPC]</name>\n  <code>C635394</code>\n  <id>6353 ...
```

```r
d1 <-
    d %>%
    as_list() %>%
    tibble (c= .)

d1 %>% head()
```

A tibble: 1 × 1

c <list>
<list [48,972]>

1 row

Before moving ahead with this process, let us have a code snippet of the xml tag that we are dealing with - which is - concept. For this, we use the htmltidy:

```{r}
htmltidy::xml_view(dconcept[1])
```

```
<concept>
    <namespace>MED-RT</namespace>
    <name>1-Compartment [PK]</name>
    <code>C298</code>
    <id>299</id>
    <status>ACTIVE</status>
    <synonym>
        <namespace>MED-RT</namespace>
        <name>Preferred Term</name>
        <to_namespace>MED-RT</to_namespace>
        <to_name>1-Compartment</to_name>
        <to_code>T82</to_code>
        <preferred>true</preferred>
    </synonym>
    <property>
        <content>MED-RT</content>
        <name>CTY</name>
        <value>PK</value>
    </property>
    <property>
        <content>MED-RT</content>
        <name>NUI</name>
        <value>N0000000060</value>
    </property>
</concept>
```

Now having converted these tags into one whole column, we need to make this one column called ‘c’ elements into different columns that correspond with the the children tag.

```{r}
d1 <-
    dconcept %>%
    as_list() %>%
    tibble (c= .)

d1 %>% head()
```

A tibble: 6 × 1

c <list>
<list [8]>
<list [29]>
<list [8]>
<list [8]>
<list [15]>
<list [12]>

6 rows

Previously, we had one colum with just one list. Now we have more number of elements to that column. To actually see what the contents of these lists are, we use unnest_wider and specifically unnest_wider on the column ‘c’ (note : when you use unnest_wider with ‘col = everything’, it will unnest every column in the tible. Thererfore when the only column in ‘c’, ‘col=everything( )’ is same as ‘col=c.’

```{r}
d2 <- d1 %>% unnest_wider(col=everything)
d2 %>% head()
```

A tibble: 6 × 2,079

namespace <list>	name <list>	code <list>	id <list>	status <list>	synonym <list>	property...7 <list>	property...8 <list>	association...9 <list>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [6]>	<list [3]>	<list [3]>	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [6]>	<list [3]>	<list [3]>	<list [7]>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [6]>	<list [3]>	<list [3]>	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [6]>	<list [3]>	<list [3]>	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<NULL>	<NULL>	<NULL>	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [6]>	<list [3]>	<list [3]>	<list [7]>

6 rows | 1-9 of 2079 columns

We see that this has association as well, but association is a different sub tag and we are not looking at association. We are only interested in concept. So in the below code, we take into consideration the columns starting with ‘property’ tag from concept.

So when we say select (namespace : status,starts_with(‘property’)), it selects all columns between namespace and status including namespace and status, and then columns starting with ‘property’. So in the below screenshot, we have gotten rid if the association tag.

```{r}
dc0<-

 dconcept %>%
 as_list() %>%
 tibble(c = .) %>%
 unnest_wider(col=c) %>%
 select(namespace:status, starts_with('property')) %>%
 pivot_longer(starts_with('property'), names_to = 'propertyname', values_to = 'property')

dc0
```

namespace <list>	name <list>	code <list>	id <list>	status <list>	propertyname <chr>	property <list>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...7	<list [3]>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...8	<list [3]>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...10	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...11	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...9	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...12	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...13	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...14	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...15	<NULL>
<list [1]>	<list [1]>	<list [1]>	<list [1]>	<list [1]>	property...17	<NULL>

In the below screenshot, we further unnest all the columns, drop stuff that have ‘NA’ in them (filter(! is.na(property_id)), unselect columns that start with id, and unnest property again. We finally get something like this :

```r
dc1<-

  dc0 %>% unnest_longer(col = property) %>%

  select(-propertyname) %>%
  filter(!is.na(property_id)) %>%
  pivot_wider(names_from = property_id, values_from = property, names_prefix = 'property_')  %>%
   unnest_longer(col = everything()) %>%
  select(-ends_with('id')) %>%
  unnest(col = starts_with('property')  ) %>%
  unnest(col = starts_with('property'))

dc1
```

R Console | tbl_df 6978 x 7

A tibble: 6,978 × 7

namespace <chr>	name <chr>	code <chr>	status <chr>	property_content <chr>	property_name <chr>	property_value <chr>
MED-RT	1-Compartment [PK]	C298	ACTIVE	MED-RT	CTY	PK
MED-RT	1-Compartment [PK]	C298	ACTIVE	MED-RT	NUI	N0000000060
MED-RT	14-alpha Demethylase Inhibitors [MoA]	C300411	ACTIVE	MED-RT	CTY	MoA
MED-RT	14-alpha Demethylase Inhibitors [MoA]	C300411	ACTIVE	MED-RT	NUI	N0000020025
MED-RT	2-Compartment [PK]	C300	ACTIVE	MED-RT	CTY	PK
MED-RT	2-Compartment [PK]	C300	ACTIVE	MED-RT	NUI	N0000000061
MED-RT	3-Compartment [PK]	C302	ACTIVE	MED-RT	CTY	PK
MED-RT	3-Compartment [PK]	C302	ACTIVE	MED-RT	NUI	N0000000062
MED-RT	4-Hydroxyphenyl-Pyruvate Dioxygenase Inhibitor [EPC]	C635228	ACTIVE	MED-RT	CTY	EPC
MED-RT	4-Hydroxyphenyl-Pyruvate Dioxygenase Inhibitor [EPC]	C635228	ACTIVE	MED-RT	NUI	N0000175809

In the above screenshot, we observe a pattern which is, CTY and NUI keep repeating in property name with their corresponding values in the property_value row. So now, we can just pivot wider ‘property_name’ with values from ‘property_value’ (meaning that CTY and NUI will be the new columns replacing the original column they are from - property_name, and the values that these two columns will have will be taken from the column ‘property value’).

A tibble: 6 × 7

namespace <chr>	name <chr>	code <chr>	status <chr>	property_content <chr>	CTY <chr>	NUI <chr>
MED-RT	1-Compartment [PK]	C298	ACTIVE	MED-RT	PK	N0000000060
MED-RT	14-alpha Demethylase Inhibitors [MoA]	C300411	ACTIVE	MED-RT	MoA	N0000020025
MED-RT	2-Compartment [PK]	C300	ACTIVE	MED-RT	PK	N0000000061
MED-RT	3-Compartment [PK]	C302	ACTIVE	MED-RT	PK	N0000000062
MED-RT	4-Hydroxyphenyl-Pyruvate Dioxygenase Inhibitor [EPC]	C635228	ACTIVE	MED-RT	EPC	N0000175809
MED-RT	5-Lipoxygenase Inhibitor [EPC]	C635394	ACTIVE	MED-RT	EPC	N0000175956

Do you rememeber when we were first looking at the concept tag in general? the concept tag had some different number of children tags ‘property and synonym’ that were repeated different number of times for every concept tag. Not all tags were different in number for all the concept tag - there were a few tags that were standard - namespace, name, code and status. So we have to figure out a solution for this issue:

<concept>
	<property>
	<synonym>
</concept>

In this case, we cannot really make all concepts into one table because property and synonym are not the same number in different concept tags. So, to solve this problem, we make concept - property into one table and concept - synonym into a different table.

The above data manipulation we have done is completely for **concept properties**.

Concept and another tag called association were one of the most challenging tags to deal with, but after a lot of exploration, we could convert the whole xml document into different csv files - which brings me to the point : Now that we have a perfect tible/ table : we need to convert this into a CSV file.

A tibble: 6 × 7

namespace	name	code	status	property_content	CTY	NUI
MED-RT	1-Compartment [PK]	C298	ACTIVE	MED-RT	PK	N0000000060
MED-RT	14-alpha Demethylase Inhibitors [MoA]	C300411	ACTIVE	MED-RT	MoA	N0000020025
MED-RT	2-Compartment [PK]	C300	ACTIVE	MED-RT	PK	N0000000061
MED-RT	3-Compartment [PK]	C302	ACTIVE	MED-RT	PK	N0000000062
MED-RT	4-Hydroxyphenyl-Pyruvate Dioxygenase Inhibitor [EPC]	C635228	ACTIVE	MED-RT	EPC	N0000175809
MED-RT	5-Lipoxygenase Inhibitor [EPC]	C635394	ACTIVE	MED-RT	EPC	N0000175956

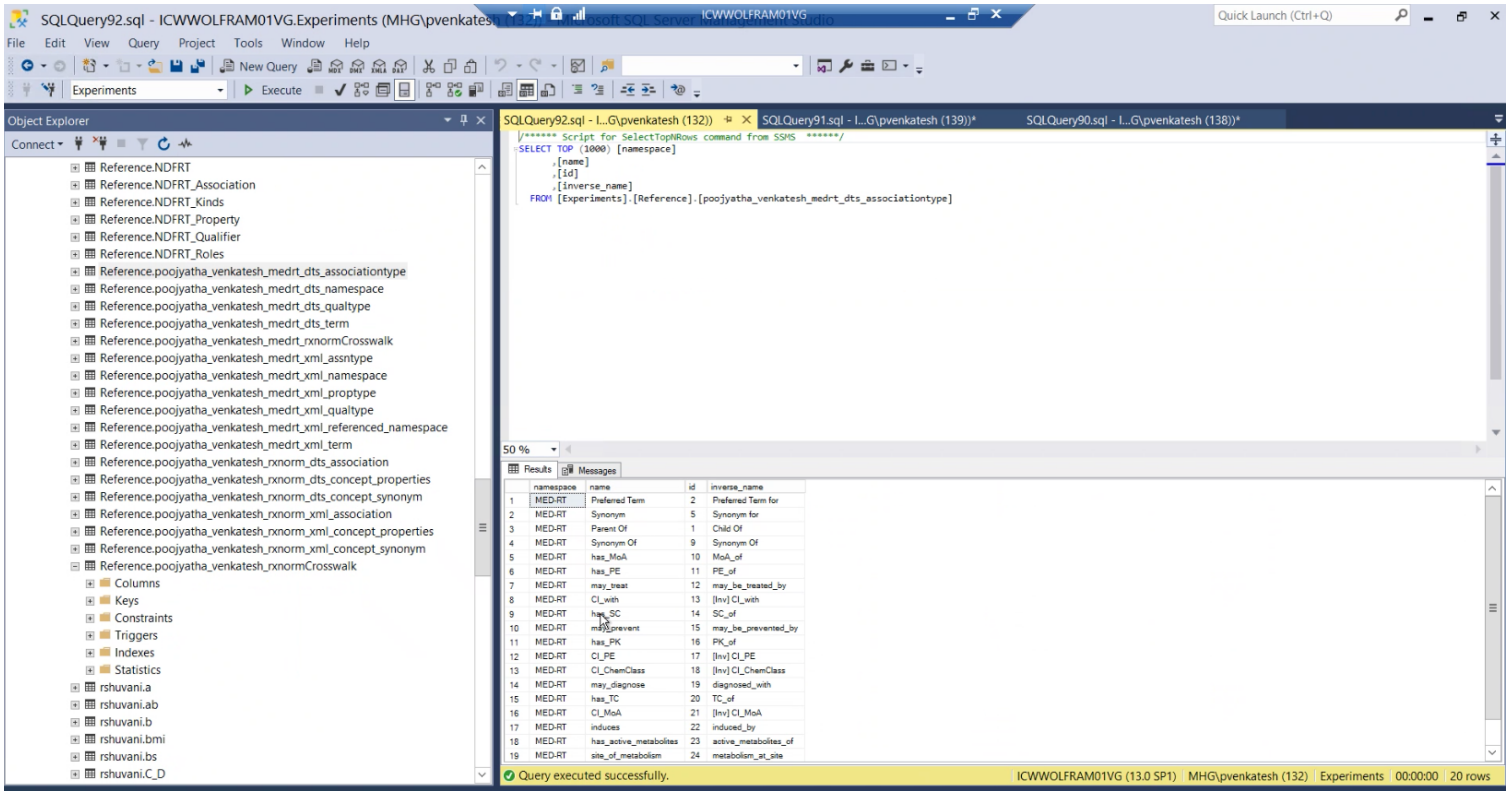
6 rows

```{r}
dc2 %>% data.table::fwrite(here::here('data','dts_concept_properties.csv'))
```

```{r}
dc2 <-  data.table::fread(here::here('data','dts_concept_properties.csv'))
```

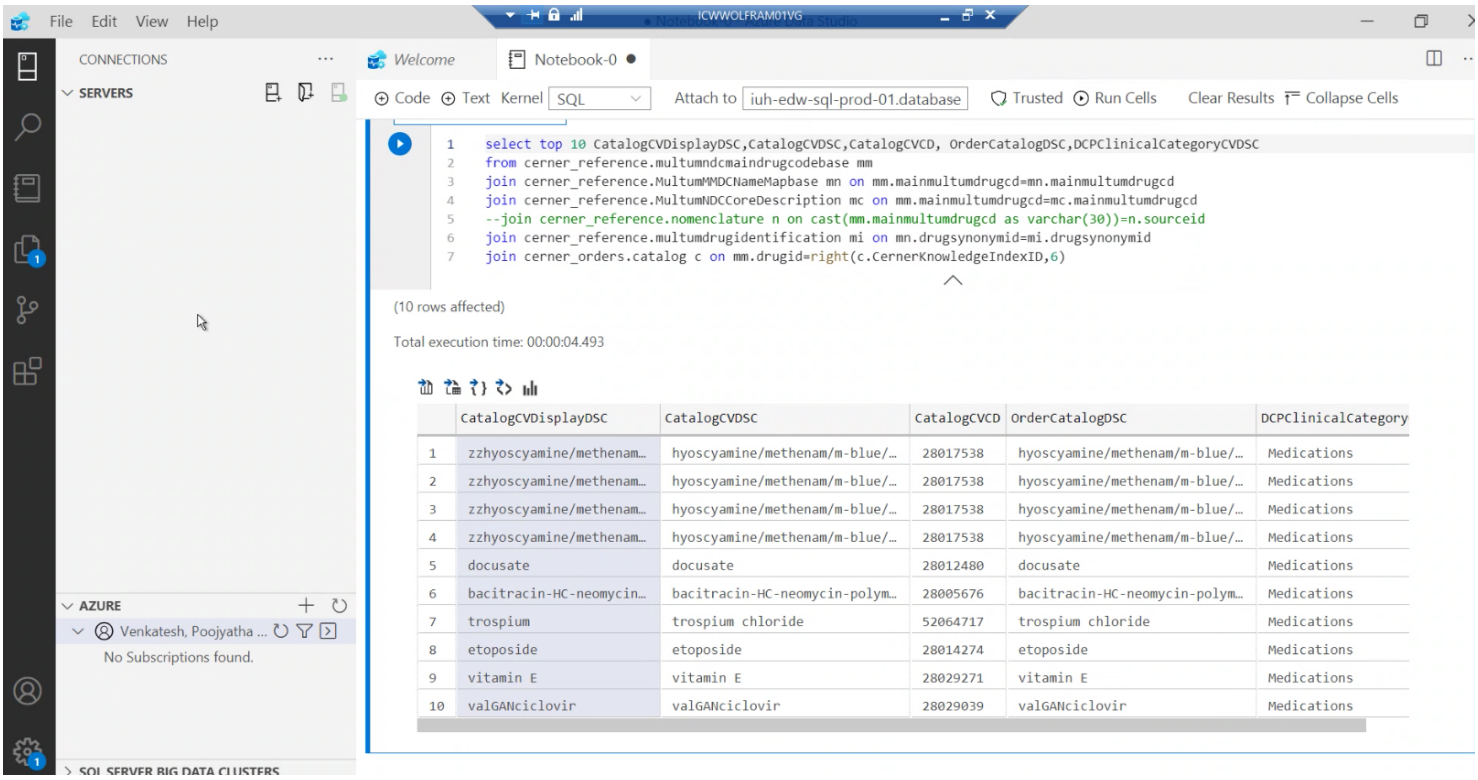
The above screenshot saves the tible as ‘dts_concept_properties.csv’.

So after converting all these xml tags into csv files, I uploaded these to Microsoft SQL Server in a folder calles ‘experiments’.



All the tables that start with ‘poojyatha_venkatesh_ …’ are the csv files that were converted from XML documents.

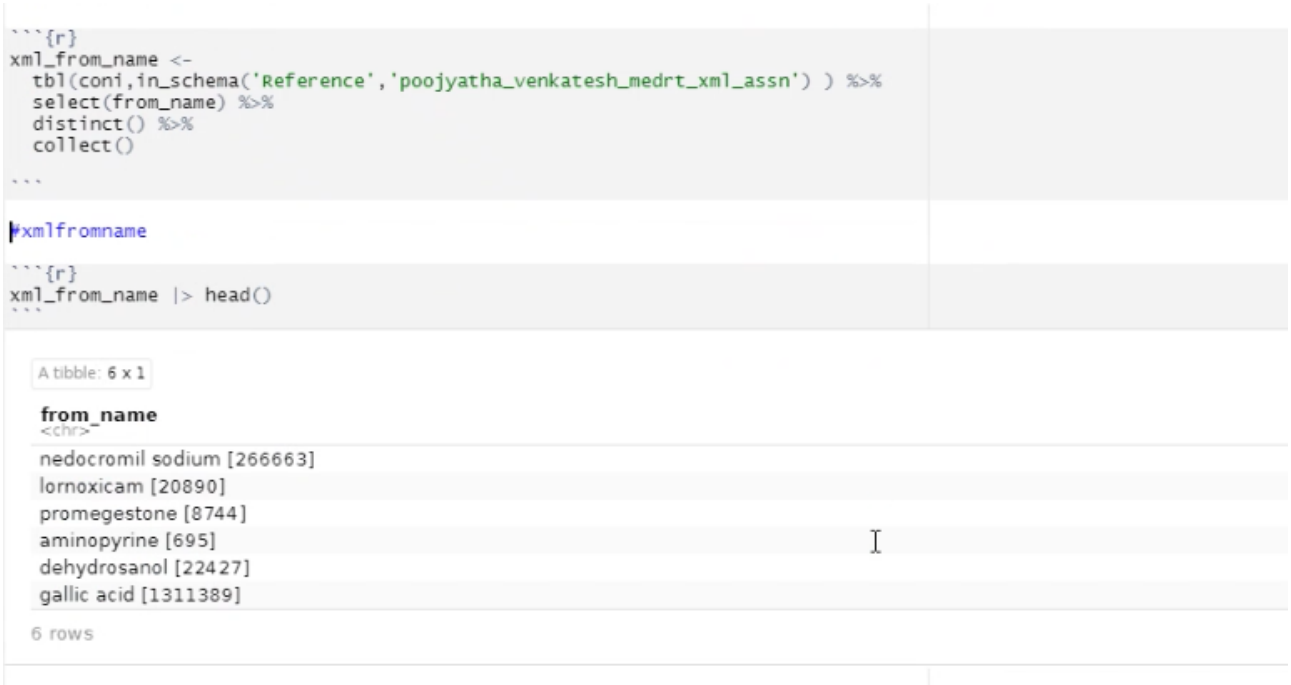
The next step was to access the data from EDW cerner or the patient data. With the help of people in IU Health, we could do it :



The next step was to find out commonalities between this data (data that was already existing in the IU Health server) and the data on MSS (microsoft sql server , in other words, that data that we uploaded).

Upon some exploration, I found that CatalogCVDisplayDSC column from the above screenshot and a column called from_name from one of the uplaoded tables called association were the same. So, we tried to create a crosswalk between the both. Below is the process of creating a crosswalk.

1. First access the table ‘association’ from MSS and then select the desired column ‘from_name’



We had to get rid of the unnecessary numbers in the abive screenshot because the values in the column ‘from_name’ essentially were the drug names. So we did that here and selected distinct drug names :

```{r}
dts_from_name <-
  tbl(coni,in_schema('Reference','poojyatha_venkatesh_medrt_dts_association') ) %>%
  select(from_name) %>%
  distinct() %>%
  collect() %>%
  mutate(from_name = stringr::str_remove(from_name, regex('\\[\\d*\\]'))) %>%
  distinct()
```

```{r}
dts_from_name |> head()
```

A tibble: 6 x 1

from_name <chr>
lornoxicam
nedocromil sodium
promegestone
aminopyrine
dehydrosanol
gallic acid

6 rows

Lets pause this process a bit.

Now to create a crosswalk, we needed to access the cerner data and bring it here. That is what we do below :

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
pacman::p_load(tidyverse,dbplyr, stringdist)
```

```{r, include=FALSE, eval=TRUE}
coni <- DBI::dbConnect(odbc::odbc(),
                 Driver = "SQL Server",
                 Server = "icwwolfram01vg",
                 Database = "Experiments")
```

```{r connection, cache=FALSE}
# The connection to CERNER data.
con <- odbc::dbConnect(odbc::odbc(), "EDW", uid = 'pvenkatesh@iuhealth.org')
```

```{r}
dn <- tbl(con,in_schema('cerner_reference','multumndcmaindrugcodebase'))
```

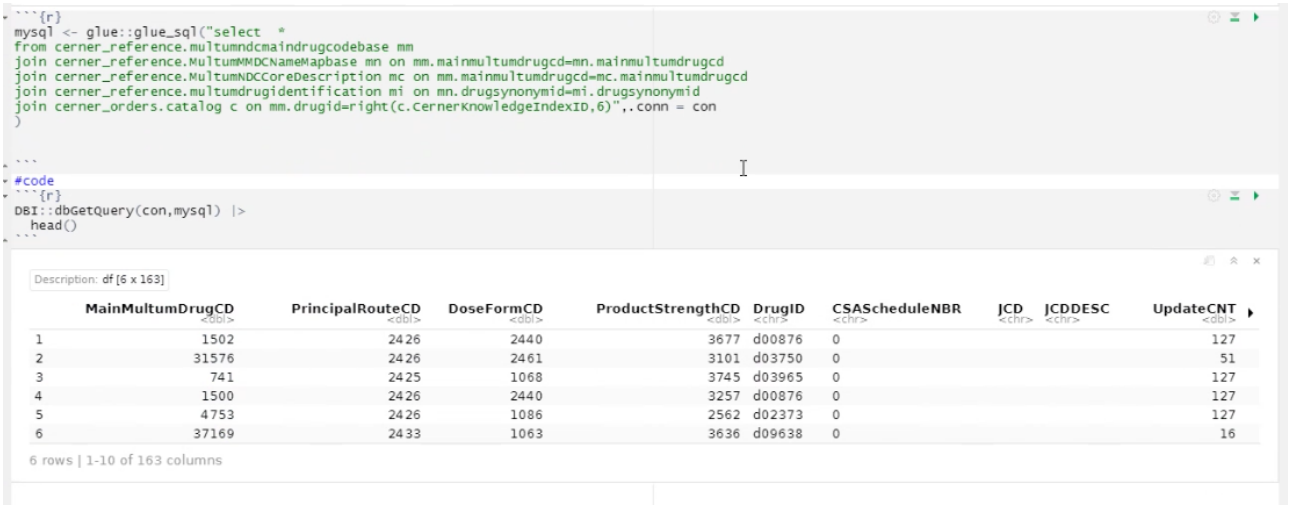
```{r}
head(dn)
```

Source: SQL [?? x 19] Database: Microsoft SQL Server 12.00.2531[pvenkatesh@iuhealth.org@iuh-edw-sql-prod-01/EDW]

MainMultumDrugCD <dbl>	PrincipalRouteCD <dbl>	DoseFormCD <dbl>	ProductStrengthCD <dbl>	DrugID <chr>	CSAScheduleNBR <chr>	JCD <chr>	JCDDESC <chr>	UpdateCNT <dbl>
27118	2420	2466	3677	d07799	0	J9042	1 mg, IJ	103
27124	2426	2461	4112	d05877	0			103
27129	2426	2440	3273	d07800	0			103
27143	2409	2460	2809	d01387	0	J0270	Per 1.25 mcg, IJ	103
27150	2409	1086	28117	d07687	2			103
24501	2426	2467	25883	d00427	0			120

6 rows | 1-9 of 19 columns

We further implement a SQL query that was given to us from someone working at IU Health :



```r
mysql <- glue::glue_sql("select  *
from cerner_reference.multumndcmaindrugcodebase mm
join cerner_reference.MultumMMDCNameMapbase mn on mm.mainmultumdrugcd=mn.mainmultumdrugcd
join cerner_reference.MultumNDCCoreDescription mc on mm.mainmultumdrugcd=mc.mainmultumdrugcd
join cerner_reference.multumdrugidentification mi on mn.drugsynonymid=mi.drugsynonymid
join cerner_orders.catalog c on mm.drugid=right(c.CernerKnowledgeIndexID,6)",.conn = con
)
```

```
#code
```

```r
DBI::dbGetQuery(con,mysql) |>
  head()
```

Description: df [6 x 163]

	MainMultumDrugCD <dbl>	PrincipalRouteCD <dbl>	DoseFormCD <dbl>	ProductStrengthCD <dbl>	DrugID <chr>	CSAScheduleNBR <chr>	JCD <chr>	JCDDESC <chr>	UpdateCNT <dbl>
1	1502	2426	2440	3677	d00876	0			127
2	31576	2426	2461	3101	d03750	0			51
3	741	2425	1068	3745	d03965	0			127
4	1500	2426	2440	3257	d00876	0			127
5	4753	2426	1086	2562	d02373	0			127
6	37169	2433	1063	3636	d09638	0			16

6 rows | 1-10 of 163 columns

But we need only the required columns and not all columns. So we create a table with only desired colmuns :

```r
#twofields included in table1
```

```r
table1 |> head()
```

Description: df [6 x 2]

	CatalogCVCD <dbl>	CatalogCVDisplayDSC <chr>
1	28009909	clomiPRAMINE
2	28011187	dexbrompheniramine
3	28005728	bacitracin-polymyxin B ophthalmic
4	28024222	phosphorated carbohydrate solution
5	8715447297	salicylic acid-sulfacetamide sodium topi
6	28028157	tiZANidine

6 rows

```r
table1 |>
  select(CatalogCVDisplayDSC) |>
  distinct() |>
  head()
```

Description: df [6 x 1]

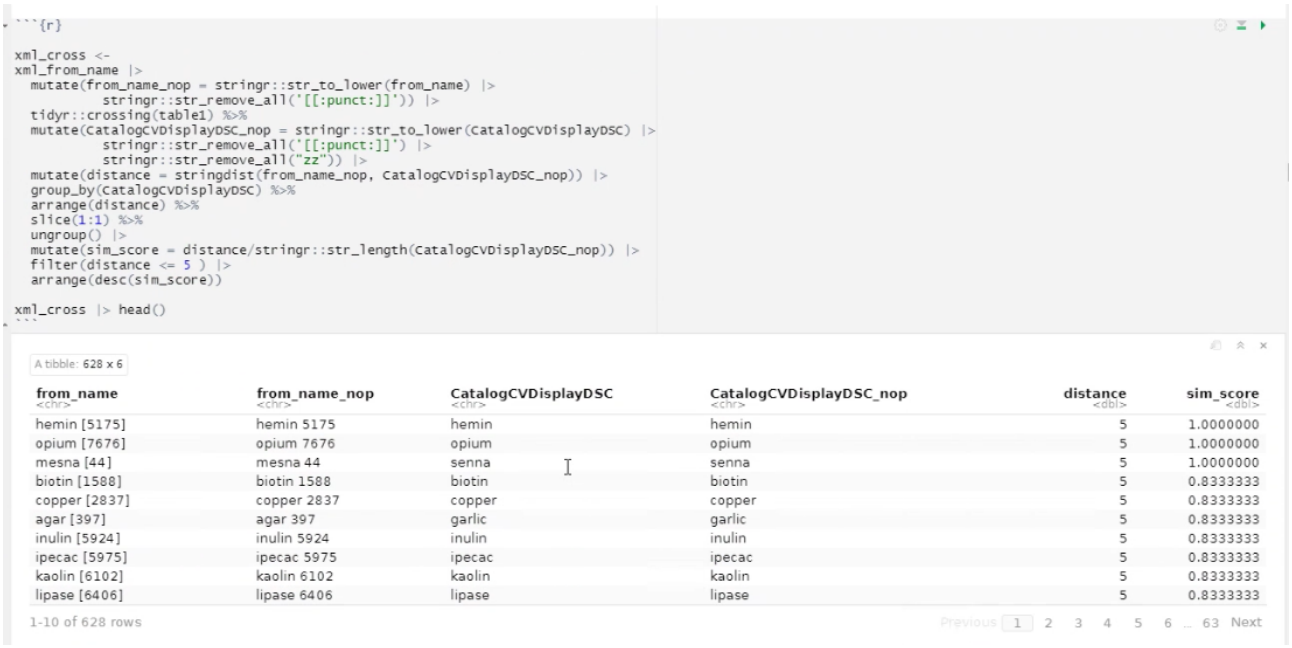
	CatalogCVDisplayDSC <chr>
1	fentaNYL-bupivacaine
2	dextran, high molecular weight
3	PHYSostigmine
4	PHENYLephrine topical
5	droxidopa
6	lisinopril

6 rows

So there we have ‘ table1’.

Now the actual crosswalk :

A crosswalk is formed by columns from table 1 and from xml association. We need to identify what exactly is same from from_name and CatalogCVDisplayDSC. For this we need to use some regular expression to convert all strings to one format without any disturbances : convert all strings to lower case or upper case (we chose lower case here), get rid of characters that add no meaning (for example we found ‘zz’ , ‘[23434]’ etc attached to a few drug names. After some cleaning, we used something calles ‘stringdist’ that calculated how close two strings are. If the string distance = 0, they are exactly the same. The more the string distance, the more they are different from eachother. But still, a string distance wouldn’t be the best way to judge if to strings are same / similar because they could be of different lengths. Therefore to normalise this, we calculate soemthing calles ‘sim_score’ to divide the dtring distance by the string length. And then by observing the sim_score, we filter rows that have a really large sim score which would mean that the strings do not match. The below screenshot denotes this process :



```r
xml_cross <-
xml_from_name |>
  mutate(from_name_nop = stringr::str_to_lower(from_name) |>
           stringr::str_remove_all('[[:punct:]]')) |>
  tidyr::crossing(table1) %>%
  mutate(CatalogCVDisplayDSC_nop = stringr::str_to_lower(CatalogCVDisplayDSC) |>
           stringr::str_remove_all('[[:punct:]]') |>
           stringr::str_remove_all("zz")) |>
  mutate(distance = stringdist(from_name_nop, CatalogCVDisplayDSC_nop)) |>
  group_by(CatalogCVDisplayDSC) %>%
  arrange(distance) %>%
  slice(1:1) %>%
  ungroup() |>
  mutate(sim_score = distance/stringr::str_length(CatalogCVDisplayDSC_nop)) |>
  filter(distance <= 5 ) |>
  arrange(desc(sim_score))

xml_cross |> head()
```

A tibble: 628 x 6

from_name <chr>	from_name_nop <chr>	CatalogCVDisplayDSC <chr>	CatalogCVDisplayDSC_nop <chr>	distance <dbl>	sim_score <dbl>
hemin [5175]	hemin 5175	hemin	hemin	5	1.0000000
opium [7676]	opium 7676	opium	opium	5	1.0000000
mesna [44]	mesna 44	senna	senna	5	1.0000000
biotin [1588]	biotin 1588	biotin	biotin	5	0.8333333
copper [2837]	copper 2837	copper	copper	5	0.8333333
agar [397]	agar 397	garlic	garlic	5	0.8333333
inulin [5924]	inulin 5924	inulin	inulin	5	0.8333333
ipecac [5975]	ipecac 5975	ipecac	ipecac	5	0.8333333
kaolin [6102]	kaolin 6102	kaolin	kaolin	5	0.8333333
lipase [6406]	lipase 6406	lipase	lipase	5	0.8333333

1-10 of 628 rows Previous 1 2 3 4 5 6 ... 63 Next

To find good matching string, we filter ones with sim_score less than 0.8571429 :

xcross

```{r}
xcross <-
  xml_from_name |>
  mutate(from_name_nop = stringr::str_to_lower(from_name) |>
           stringr::str_remove_all('[[:punct:]]')) |>
  tidyr::crossing(table1) %>%
  mutate(CatalogCVDisplayDSC_nop = stringr::str_to_lower(CatalogCVDisplayDSC) |>
           stringr::str_remove_all('[[:punct:]]') |>
           stringr::str_remove_all("zz")) |>
  mutate(distance = stringdist(from_name_nop, CatalogCVDisplayDSC_nop)) |>
  group_by(CatalogCVDisplayDSC) %>%
  arrange(distance) %>%
  slice(1:1) %>%
  ungroup() |>
  mutate(sim_score = distance/stringr::str_length(CatalogCVDisplayDSC_nop)) |>
  filter(sim_score < 0.8571429 ) |>
  arrange((sim_score))

xcross
```

A tibble: 3,392 x 6

from_name <chr>	from_name_nop <chr>	catalogCVDisplayDSC <chr>
chorionic gonadotropin [4986]	chorionic gonadotropin 4986	chorionic gonadotropin (HCG)
5-hydroxytryptophan [94]	5hydroxytryptophan 94	5-hydroxytryptophan
dihydroxyaluminum sodium carbonate [23163]	dihydroxyaluminum sodium carbonate 23163	zzdihydroxyaluminum sodium carbonate
aluminum chloride hexahydrate [17611]	aluminum chloride hexahydrate 17611	aluminum chloride hexahydrate topical
colistin sulfate [2710]	colistin sulfate 2710	colistin sulfate otic
edetate disodium [3539]	edetate disodium 3539	edetate disodium (EDTA)
cytomegalovirus immune globulin [22178]	cytomegalovirus immune globulin 22178	cytomegalovirus immune globulin
collagenase Clostridium histolyticum [898492]	collagenase clostridium histolyticum 898492	collagenase clostridium histolyticum
chlorophyllin copper complex [82242]	chlorophyllin copper complex 82242	zzchlorophyllin copper complex topical
Cholera Vaccine [2427]	cholera vaccine 2427	cholera vaccine, live

The final crosswalk would contain from_name from association from bith xml as well as dts. So the final crosswalk is from :

xcross (xml_association) X dcross (dts_association) X table1 (cerner data)

```{r}
#3
```{r}
finalcw <-

  xcrosswalk |>
  bind_rows(dcrosswalk) |>
  arrange(distance)


finalcw |> head()

```
```

For example the above screenshots depicts crosswalk between xml crosswalk and the dts cross walk. (xcross is already between xml (xml association) and table 1 (cerner data)).

Therefore, the final crosswalk is below :

```r
finalcw <-
finalcw |>
  select(CatalogCVCD, CatalogCVDisplayDSC,from_name)


finalcw
```

A tibble: 43,500 x 3

CatalogCVCD <dbl>	CatalogCVDisplayDSC <chr>	from_name <chr>
28002755	5-hydroxytryptophan	5-hydroxytryptophan
28002757	abacavir	abacavir
5316604749	abaloparatide	abaloparatide
149479389	abatacept	abatacept
5796401533	abemaciclib	abemaciclib
881079243	abiraterone	abiraterone
758263483	abobotulinumtoxinA	abobotulinumtoxinA
10004706639	abrocitinib	abrocitinib
5796401651	acalabrutinib	acalabrutinib
61973209	acamprosate	acamprosate

1-10 of 43,500 rows

Next we upload this to MSS similar to how we did the csv files before.

#upload to experiments

```r
data.table::fwrite(finalcw,here::here('data','finalcw.csv'))
```

```r
rxnormCrosswalk <- data.table::fread(here::here('data','finalcw.csv'))
```

```r
coni <- DBI::dbConnect(odbc::odbc(),
                 Driver = "SQL Server",
                 Server = "icwwolfram01vg",
                 Database = "Experiments")


```

Now that everything is in MSS, our task is to identify groupings. For this, we need to find a criteria for grouping drugs. We wanted to first find if we can find which drugs (from_name from corsswalk or xml_association / dts_association tables) belong to which concept type (CTY form concept properties table) Upon analysing : we found something interesting :

Crosswalk :

SQLQuery97.sql - I...G\pvenkatesh (142))* SQLQuery96.sql - I...G\pvenkatesh (141))*

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [CatalogCVCD]
      ,[CatalogCVDisplayDSC]
      ,[from_name]
  FROM [Experiments].[Reference].[poojyatha_venkatesh_rxnormCrosswalk]
  WHERE from_name like '%amino%'
```

50 %

Results | Messages

	CatalogCVCD	CatalogCVDisplayDSC	from_name
1	28002791	acetaminophen	acetaminophen
2	28002791	acetaminophen	acetaminophen
3	28002791	acetaminophen	acetaminophen
4	28002791	acetaminophen	acetaminophen
5	28002791	acetaminophen	acetaminophen
6	28002791	acetaminophen	acetaminophen
7	28002791	acetaminophen	acetaminophen
8	28002791	acetaminophen	acetaminophen
9	28002791	acetaminophen	acetaminophen
10	6786735697	aminolevulinic acid	aminolevulinic acid
11	6786735697	aminolevulinic acid	aminolevulinic acid

Concept properties :

SQLQuery98.sql - I...G\pvenkatesh (143))* SQLQuery97.sql - I...G\pvenkatesh (142))* SQLQuery96.sql - I...G\pvenkate

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [namespace]
      ,[name]
      ,[code]
      ,[status]
      ,[property_namespace]
      ,[CTY]
      ,[NUI]
  FROM [Experiments].[Reference].[poojyatha_venkatesh_rxnorm_xml_concept_properties]
  WHERE name like '%amino%'
```

50 %

Results | Messages

	namespace	name	code	status	property_namespace	CTY	NUI
1	MED-RT	Para-Aminobenzoic Acid Inhibitors	N0000009946	A	MED-RT	MoA	N0000009946
2	MED-RT	gamma-Aminobutyric Acid-ergic Agonist	N0000175759	A	MED-RT	EPC	N0000175759
3	MED-RT	gamma-Aminobutyric Acid A Receptor Agonist	N0000183360	A	MED-RT	EPC	N0000183360
4	MED-RT	Neuroactive Steroid Gamma-Aminobutyric Acid A Re...	N0000194006	A	MED-RT	EPC	N0000194006
5	MED-RT	Aminoketone	N0000180855	A	MED-RT	EPC	N0000180855
6	MED-RT	Aminoglycoside Antibacterial	N0000175477	A	MED-RT	EPC	N0000175477
7	MED-RT	Antidote for Acetaminophen Overdose	N0000175961	A	MED-RT	EPC	N0000175961
8	MED-RT	Aromatic Amino Acid Decarboxylation Inhibitor	N0000175754	A	MED-RT	EPC	N0000175754
9	MED-RT	Glycosaminoglycan	N0000175837	A	MED-RT	EPC	N0000175837
10	MED-RT	Amino Acid	N0000175780	A	MED-RT	EPC	N0000175780
11	MED-RT	Aromatic Amino Acid	N0000193220	A	MED-RT	EPC	N0000193220
12	MED-RT	Amino Acid Hypertonic Solution	N0000175797	A	MED-RT	EPC	N0000175797

xml_term table :

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [namespace]
      ,[name]
      ,[code]
      ,[id]
      ,[status]
  FROM [Experiments].[Reference].[poojyatha_venkatesh_medrt_xml_term]
  WHERE name like '%amino%'
```

50 %

Results | Messages

	namespace	name	code	id	status
1	MED-RT	Acetaminophen	T1654	1654	ACTIVE
2	MED-RT	Acetaminophen-containing Agent	T1653	1653	ACTIVE
3	MED-RT	Amino Acid	T3212	3212	ACTIVE
4	MED-RT	Amino Acid Hypertonic Solution	T3149	3149	ACTIVE
5	MED-RT	Aminoglycoside	T3285	3285	ACTIVE
6	MED-RT	Aminoglycoside Antibacterial	T3282	3282	ACTIVE
7	MED-RT	Aminoglycoside Antimicrobial	T3283	3283	ACTIVE

We found some connection and Concept Types that had “amino” in them. This connection can be further explored to see what drugs belong to what concept types. The table term is acting as an intermediate between crosswalk and association tables.

Drugs can also be grouped on their own based on certain characteristics. For example, in the below screenshot, I am grouping drugs based on ‘Alcohol Drinking’.

SQLQuery99.sql - I...G\pvenkatesh (145))* | SQLQuery98.sql - I...G\pvenkatesh (143))* | SQLQuery97.sql - I...G\pvenkatesh (142))*

```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP (1000) [namespace]
      ,[association_name]
      ,[from_namespace]
      ,[from_name]
      ,[to_namespace]
      ,[to_name]
      ,[qualifier_content]
      ,[qualifier_name]
      ,[qualifier_value]
      ,[from_code]
      ,[to_code]
  FROM [Experiments].[Reference].[poojyatha_venkatesh_rxnorm_dts_association]
  WHERE to_name='Alcohol Drinking'
```

50 %

Results | Messages

	namespace	association_name	from_namespace	from_name	to_namespace	to_name	qualifier_content	qualifier_name	qualifier_value	from_code	to_code
1	MED-RT	CI_with	RxNorm R	benznidazole	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	18994	M0000650
2	MED-RT	CI_with	RxNorm R	flibanserin	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	1665509	M0000650
3	MED-RT	CI_ChemClass	RxNorm R	calcium oxybate	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	2387307	M0000650
4	MED-RT	CI_ChemClass	RxNorm R	magnesium oxybate	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	2387308	M0000650
5	MED-RT	CI_ChemClass	RxNorm R	oxybate	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	2387302	M0000650
6	MED-RT	CI_ChemClass	RxNorm R	potassium oxybate	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	2387309	M0000650
7	MED-RT	CI_ChemClass	RxNorm R	sodium oxybate	MeSH	Alcohol Drinking	MED-RT	Authority	MEDRT	9899	M0000650

Please note that we are dealing with datasets from two different groups : XML and DTS. They have similar base tags but there are some differences in these tags as well.