**Pfizer readme p\_meta D3**

Study specifications as described in the SAP are translated to a set of tables in csv format. This set of tables is stored in the folder <https://github.com/UMC-Utrecht-RWE/Pfizer/tree/Interim-3-development/Data%20characterisation/PfizerScript/p_meta_data>.

If there are changes in the specifications most of these changes can be effectuated by make changes in these files. Moreover, if data is missing in the output of the script for a specific DAP, the first thing to check is if these metafiles are filled correctly. The script is importing and using the information stored in these tables.

In the p\_meta folder a lot of files are stored. However, the following files are essential (see figure below):

Needed for the translation of the SAP (in the green block in the figure at page 2)

* Pfizer\_study\_variables.csv
* Pfizer\_algorithms.csv
* Pfizer\_dictionary.csv
* Pfizer\_scores.csv
* (Pfizer AESI\_information)

Needed for extraction of the concepts/study variables the following are essential (red)

* Pfizer\_additional\_concepts.csv
* 20221208\_ALL\_full\_codelist.csv
* 20221208\_ALL\_drug\_proxies\_codelist.csv
* Pfizer\_vaccines\_codelist.csv

Also you may want to check the following file to get an overview of the steps in the script (orange)

* Program.csv

In the script first the meta files are analysed and the needed information is extracted and stored in global variables or in .rds files in the folder Pfizer\Data\characterisation\PfizerScript\g\_intermediate\tmp. This is done in the steps:

Step\_00\_SetParameters.rds

Step\_00\_SetCodeSheets.rds

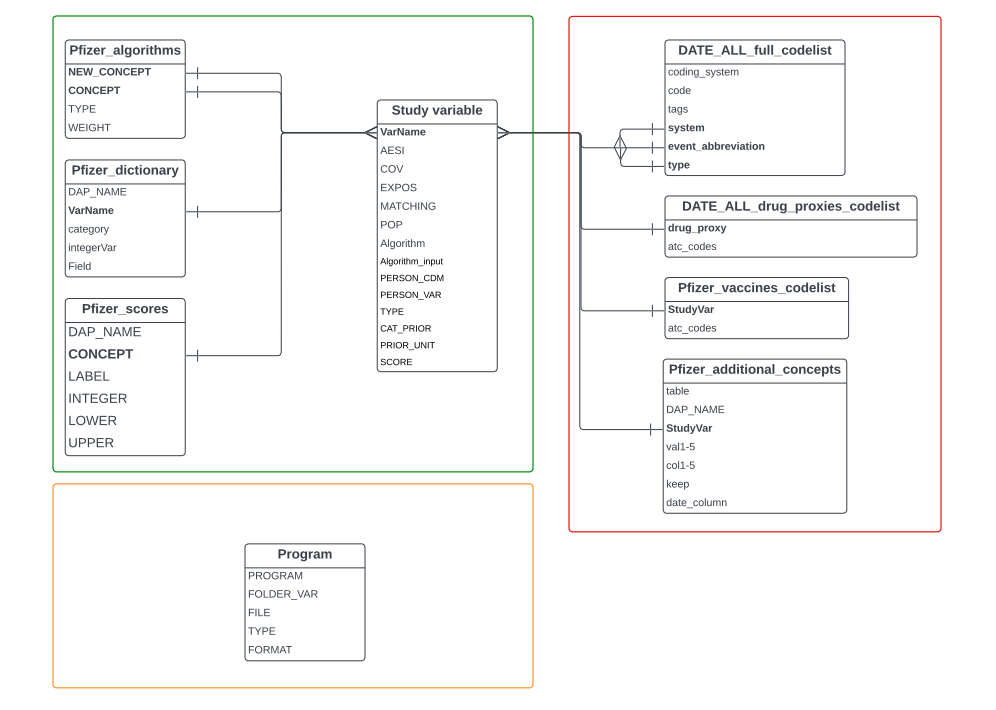
Note that when you want to run the script in multiple stages, you always first need to rerun those 2 scripts because it creates the needed variables and R objects. These are used throughout the script.

See appendix 1 for a diagram of all the steps in the script. This diagram subdivides the script in multiple stages. The visualisation is not 100% accurate because of the complexity of the process.

TODO: add a script that checks the assumptions of the metafiles. This is partly done in Step\_00\_SetParameters.rds but there are more checks that you may want to add

**Model**

All the tables that are mentioned at page 1, with the relevant variables, are shown in the figure below. The lines indicate how the tables are connected. Note that the files and related script where developed piece by piece over a long period of time. So namings and relations are not perfect. With knowledge from now it would be more structured and model would be modified accordingly. Nevertheless, the model represented here contains 80% of the study information and the D3 script is built upon this layer.



**Advise**

It is recommended to avoid using Excel when working with CSV tables from the model. Instead, it is more suitable to use Notepad or Notepad++ as saving the file in Excel can unintentionally alter the file's formatting and introduce unwanted spaces, leading zeros, or strange characters. While Excel may be more user-friendly, it is advisable to open the file in Notepad after making all the changes in Excel and review it for any spaces, strange characters, or leading zeros.

**Description of the meta tables in the model**

1. **SAP study parameters**

**1.1 Pfizer\_study\_variables.csv**

The aim of this file is defining all study variables that are within the study. These names are used to connect the other meta files and this are the column names in the output of the script. Moreover, per study variable it is defined what it’s role is within the study. Study variables can have multiple roles within the study.

Columns:

* VarName (string|key): The list of study variables. 1 VarName per row.
* AESI (TRUE/FALSE) : If TRUE this variable is a main outcome meaning that in script Step\_13\_AddAESI all first occurrences after T0 are collected and these are used to calculate indecencies.
* COV: If TRUE these variables are a covariate. In the script at Step\_12\_AddCoVariates all the most recent occurrences from T0 are collected with respect to a lookback time (also stored in the Pfizer\_study\_variables.csv files(CAT\_PRIOR)). These dates are set to 1/0. Moreover, in Step\_12c\_AddWeights these variables are used in the IPW model.
* EXPOS (TRUE/FALSE): If TRUE this variable represents the T0 for the study.
* MATCHING: (TRUE/FALSE): If TRUE then this variables is used as a matching factor. This column interacts with PERSON\_CDM and TYPE and the scripts Step\_07\_CreateSpellsMatching/ Step\_08\_PrepareExposedControlsSpellsCombined/ Step\_09\_MatchingProcedure are using this information.
* POP (TRUE/FALSE): This column is not used, but was mend to store which columns are involved in the creation of cohorts if they would be there.
* Algorithm(TRUE/FALSE): If TRUE, this are secondary study variables that are composed by other primary study variables who are directly available in the CDM. In scripts Step\_05\_GetConceptsMatching and Step\_11\_PutConceptsInDatabase the primary study variables are collected and then in scripts Step\_05\_GetConceptsMatching/ Step\_11b\_CreateAdditionalConcepts and Step\_12\_AddCoVariates these secondary study variables will be created. The information for this is stored in the connected table Pfizer\_ algorithms.
* Algorithm\_input(TRUE/FALSE): this variable indicates if it is used as an input for a secondary study variable. This variable is not needed for the script since this can be extracted from Pfizer\_ algorithms.CONCEPT. So it currently merely serves as an overview.
* PERSON\_CDM (TRUE/FALSE): This is used to indicate if a variable is a property if a person, meaning that it fits in a table with 1 row per subject id. This is used to extract study variables directly from an intermediate persons table (PERSONS2.rds) via PERSON\_VAR. In variables where MATCHING is TRUE this indicates if a variable is time depended (FALSE -> spell files created in Step\_07\_CreateSpellsMatching) or time independent (TRUE > added to the matching database in Step\_08\_PrepareExposedControlsSpellsCombined).
* PERSON\_VAR (string): here the column name in the PERSONS.rds is stored. In some cases this is similar to the CDM persons table column name. However, some variables are created in the script Step\_01\_CreatePersons because they are not available directly in the CDM.
* TYPE (TF/CAT/SUM if COV = TRUE): this variable indicates if a covariate is a binary variable, in which the date is collected, or it is a categorical variable in which also a value is collected. In scripts Step\_05\_GetConceptsMatching and Step\_11\_PutConceptsInDatabase the concepts.db is created. In TF variables the Value column is filled with the code needed for extraction. This value is not relevant and in Step\_12\_AddCoVariates the value is filled with 1 if a record is found and 0 if not. SUM is a special one and is added later. This is a TF variable with the difference that in Step\_12\_AddCoVariates not the most recent to T0 is stored. Instead the number of rows is counted and stored. SUM and CAT can only be in combination with COV = TRUE and AESI = FALSE. TF can be with both TRUE.
* CAT\_PRIOR (CLOSE\_PRIOR\_POST/PRIOR/BETWEEN OR numerical): This column is having

multiple functions. It was first created for categorical variables. So if TYPE = CAT then it is giving information about how to extract as a covariate (COV = TRUE) in Step\_12\_AddCoVariates:

* + CLOSE\_PRIOR\_POST: this is a greedy method. First it looks back from T0. If nothing found it looks forward.
  + PRIOR: looks only back from T0 on.
  + BETWEEN: this is for variables with a start and end date (pregnancy) and collects the rows in which T0 is between the start and end date of the study variable.

Then later in the process this column was also used to for TF variables to store the lookback time used Step\_12\_AddCoVariates.

* PRIOR\_UNIT(year/day/month): If in CAT\_PRIOR a number is filled and TYPE = TF then the unit is filled that is used to set all lookback time to days.

**1.2 Pfizer\_algorithms.csv**

This file stores which primary variables are needed to create a secondary variable.

Columns:

* NEW\_CONCEPT (string): the secondary variable name, or the variable name of the stud variable that’s need to be created out of other study variables.
* CONCEPT (string): the primary study variables that are needed to create a secondary study variable.
* TYPE (OR/): If OR this means that in Step\_05\_GetConceptsMatching and Step\_11b\_CreateAdditionalConcepts the CONCEPTS for a NEW\_CONCEPTS are appended to create the new study variable. This is a simple algorithm. If a more difficult algorithm needs to be created is was the intentions that this also needs to be filled in this table but then with TYPE empty and in another column a reference to the used script and used function. This is not done in Pfizer yet. TODO: fill all the complex algorithms in this file and see what to do with the variable “hardCodedAlgorithms” that is created in Step\_00\_SetParameters.
* WEIGHT (integer): this was added later on in the process. If a weight is filled it is not treated as a TYPE = OR algorithm (also if OR is filled) but as another automated algorithm. This is done after the covariate extraction in Step\_12\_AddCoVariates while a OR algorithm is created before Step\_12\_AddCoVariates. It sums the weights of the most recent study variables needed for the secondary study variable. If all weights are 1 then is equal to a sum of the distinct extracted study variables. So if 10 prescriptions for medicine 1 and 5 for medicines 2 then the result is 2.

**1.3 Pfizer\_dictionary.csv**

The main aim of this file is to align values from categorical variables. For different databases these may differ and can be standardized via this file.

Columns:

* DAP\_NAME (string): the name of the DAP equal to the dap name used in the script. The script only continues with the rows that are equal to the DAP variable in the script after Step\_00\_SetCodeSheets.
* VarName (string): name of the study variable that is the relation with Study\_varaibles.csv
* Category (string): the standardized name that needs to be outputted by the script.
* intergerVal (integer): an integer value that is connected to the category. For performance first the integer is stored in D3 (Step\_11d\_ApplyDictionaryIntegers) and used in D4. On the report level the integer is replaced by the category. Note that per VarName the category and integerVal combination needs to be unique. Moreover, 0 is not allowed as integerVal because that is used to indicate for missingness. So if in smoking status for category the value is “current” and the oriVal 3. Then for all the DAP’s “current” needs to be 3 and 3 is allowed only in combination with “current”. TODO: You may want to make a check for this or further normalize the model.
* oriVal (string): this is the value that is used in the DAP and the value that needs to be renamed to category/intergerVal.

**1.4 Pfizer\_scores.csv**

Besides study variables that are categorical and need to have standardized value labels, there are also continues study variables that need to be translated into a categorical value. This is done via this table in a similar manner as with the Pfizer\_dictionary.csv in the scripts Step\_05\_GetConceptsMatching/ Step\_12\_AddCoVariates. Note that in an earlier stage of the study the scoring was connected with the weighting algorithm. See Pfizer\_algorithms.csv|WEIGHTS. Later it became 2 independent functionalities.

Columns:

* DAP\_NAME (string/ALL): the name of the DAP equal to the dap name used in the script. The script only continues with the rows that are equal to the DAP variable in the script after Step\_00\_SetCodeSheets.

This column was added later in the study because it occurred that some DAP’s did have categorical values for socio economic status while others had continues values. ALL is used if the scoring applies for all DAP’s.

* CONCEPT (string): name of the study variable that is the relation with Study\_varaibles.csv
* LABEL (string): the standardized string that needs to be outputted by the script if the numerical value is within the range specified with LOWER and UPPER.
* INTEGER (integer): the value that is connected to the label and is stored in the D3 and used in the D4 for performance purposes. Note also here that like in Pfizer\_dictionary.csv|integerVal the label/integer combinations needs to be unique if several DAP’s are in for the same study variable.
* LOWER (integer): the lower boundary of the range which should be > 0
* UPPER (integer): the upper boundary of the range that should be > LOWER. Note that all values that can be available in the data need to be covered and that the ranges are not allowed to overlap.

**1.5 Pfizer AESI\_information**

This file appeared in the meta files after programmers of the D4 did add it on a certain moment in time. This file overlaps in information with the files described in this SAP. TODO: include this file in the model. Now the same things are stored on 2 places. This will require some code changes for both D3 and D4 programmers.

1. **Extracting CONCEPTS from the CDM**

The information needed to create most of the study variables is spread throughout the whole CDM. Some of this information is stored in a standardized manner in known CDM tables and columns. Within the Pfizer study, vaccine, medicine and event information is stored in a common standardized manner and to extract these concepts, predefined code lists are used. Note that these lists are delivered to the programmer by the study team.

Moreover, for some study variables the needed information is stored in DAP specific manner. This information can everywhere in the CDM. Therefore, the location of this information needs to be specified very precise via the Pfizer\_additional\_concepts.csv. To ensure that this file is filled correctly is a common responsibility of the study team, programmer and the DAP.

The code lists are used in the following scripts:

* Step\_05\_GetConceptsMatching
* Step\_11\_PutConceptsInDatabase
* Step\_03\_GetVaccins (only vaccines)

As a result of these scripts the database concepts.db is created in the folder Pfizer\Data characterisation\PfizerScript\g\_intermediate\concepts. In this database for every concept a table is created with a date column (Date) and a column in which the value is stored (Value). The latter is only relevant for categorical variables. How a particular row is found in the CDM is stored in the Voc column. This Voc column can also contain important meta information such as a unit of a value. These tables are eventually the main input for the creation of most of the study variables.

* 1. **Standardized Code lists**

For the extraction of concepts via these standardized code lists the function CreateConceptDatasets.R is used. This function needs a coding\_system in combination with a code to relate a row in the CDM to a concept. It is built in to use a wildcard. This is done in 2 situations:

* The coding system is stored in the variable “start\_with\_colls” created in the script Step\_00\_SetParameters. This variable is an input for the function.
* The code is ending on a dot “code1234.”

Note also that the functions removes dots from the code before comparing codes.

**2.1.1 DATE\_ALL\_full\_codelist.csv**

The aim of this list is to extract event information from the events tables of the CDM. This is done via the combination of a coding system and a code.

Columns:

* coding\_system (string): DAP’s can have their information been stored using several coding systems. The values stored in this column need to match exact with the event\_record\_vocabulary in the events table delivered by the DAP. For PHARMO and CPRD there are some exceptions that are hard coded in the scripts Step\_05\_GetConceptsMatching/ Step\_11\_PutConceptsInDatabase/ Step\_00\_SetCodeSheets. PHARMO us using the coding system free\_text in the events table and CPRD is using the coding system MEDCODEID in EVENTS and MEDICAL\_OBSERVATIONS.
* code (string): here the code is stored.
* tags (narrow/possible/delete/excluded): this gives additional information about the interpretation of the code. The study team can decide that which tags to use and these decisions are effectuated in the script Step\_00\_SetCodeSheets. TODO: there are new tags (delete/excluded) so check if this affects the decisions made earlier.
* system/event\_abbreviation/type (string) : name of the study variable that is the relation with Study\_varaibles.csv

**2.1.2 DATE\_ALL\_drug\_proxies\_codelist.csv**

The aim of this list is to extract vaccine information from the CDM via the atc codes stored in the column medicinal\_product\_atc\_code. In contrast to the events information, there is no column with the information about the coding system. This information is in the column name. However, the function CreateConceptDatasets.R needs that column. So in Step\_00\_SetCodeSheets.R this column is added.

Columns:

* StudyVar (string): name of the study variable that is the relation with Study\_varaibles.csv
* atc\_codes (string): here the code is stored.

**2.1.3 Pfizer\_vaccines\_codelist.csv**

The aim of this list is to extract vaccine information from the CDM via the atc codes stored in the column vx\_atc. There are some differences for the extraction of vaccines compared to events and medicines. This code list is used only in the script Step\_03\_GetVaccins. Because COVID vaccination information is needed for the creation of the study population, vaccines are extracted earlier in the program and first saved as .rds files (later copied to the concetps.db).

Moreover, not all DAP’s do use atc codes. Some fill the vx\_type column. For the vaccines that are loaded via the vx\_type the Pfizer\_additional\_concepts.csv needs to be used. In Step\_00\_SetCodeSheets.R this information is added to the tmp/CODES\_VACCINES.rds and removed from the tmp/CODES\_ADDITIONAL.rds.

Like medicines, the table does not contain a variable that relates to a coding system while this is required in the function that extracts the concepts. In contrast to medicines this column is added within the script Step\_03\_GetVaccins instead of in Step\_00\_SetCodeSheets

Columns:

* StudyVar (string): name of the study variable that is the relation with Study\_varaibles.csv
* atc\_codes (string): here the code is stored.

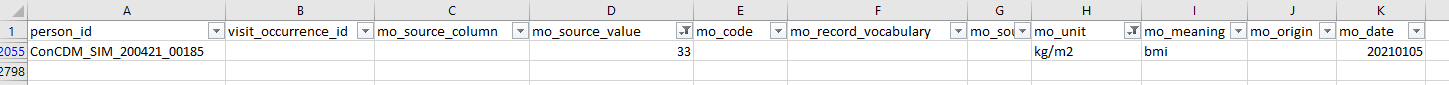
TODO: harmonize this procedure to medicines.

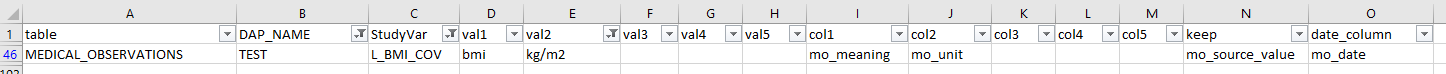
**2.2 Unstandardized information**

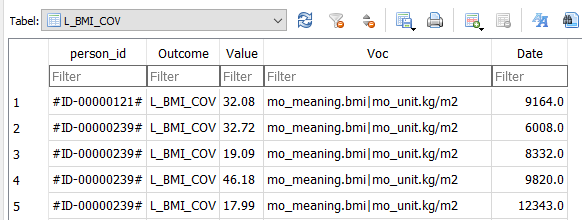
**2.2.1 Pfizer\_additional\_concepts.csv**

There are concepts that are not I the CDM via a standardized coding system/code combination.

The aim of this file is to store the information needed to find these concepts within CDM of the several DAP’s. The information this file contains is used in scripts Step\_05\_GetConceptsMatching/ Step\_11\_PutConceptsInDatabase via the function CreateConceptDatasetsMultipleVars.R. (Note that this is a different function then used for extracting the concepts via the standardized code lists.)

The figure below shows an example observation out of the MEDICAL\_OBSERVATIONS.csv of the DAP TEST. We want to extract the information needed for the study variable L\_BMI\_COV

Given this observation as example the Pfizer\_additional\_concepts.csv file will be filled like this figure below.

After the extraction of the concepts to the database it is in the concepts.db as below

Columns:

Table (string): the CDM table where the concept of interest will be found. Note that if in a DAP tables are stored with a suffix, this suffix is not relevant. Only table names known in the CDM are valid. So if MEDICAL\_OBSERVATIONS\_2021 then only MEDICAL\_OBSERVATIONS should be filled.

DAP\_NAME (string): the name of the DAP equal to the dap name used in the script. The script only continues with the rows that are equal to the DAP variable in the script after Step\_00\_SetCodeSheets.

StudyVar: name of the study variable that is the relation with Study\_varaibles.csv

val1-5 (string): this series of columns works together with col1-5 and serves as the input for filtering the correct rows from the table. Val1 needs to be filled with the value in the column that is filled in col1. So if col1 is filled with *mo\_meaning* then val1 needs to be filled with *bmi*. This is then interpreted as WHERE mo\_meaning = “bmi” is a SQL expression. If you only want to extract bmi’s of with a particular unit you need to use the val2/col2 combination. So than also fill val2 with kg/m2 and col2 with mo\_unit. Then this is interpreted as WHERE mo\_meaning = “bmi” AND mo\_unit = “kg/m2”. DAP’s are free to decide which col/val combinations do represent the wanted concept without also loading not wanted rows. However, in more complex algorithms you may need some meta information form the variable. Bmi is an example of this. The algorithm that calculates bmi is not generic and needs the information about the untit. In this situation the PI and programmer need to check if the unit is in the col/val combination or where expression. All the components of the where expression are loaded into the “Voc” column (in table L\_BMI\_COV in concepts.db ) in a standardized manner. Col and val components are separated by a dot “.” while multiple col/val combinations are separated bij “|”. Via this assumption the bmi algorithm can extract the unit form the Voc column if the filled unit is known by the algorithm.

col1-5 (string): see Val1-5

keep (string): in TF variables this needs to be left empty. If a categorical or continues variable this column is needed as a pointer to the value of interest. In the bmi example you want to have the 33 that is stored in the mo\_source\_value column.

date\_columns (string): every row is interpreted as an observation and so a date is demanded. In this column the column where the date is stored needs to be filled. In our example this will be mo\_date. In some CDM tables there are multiple date columns. In that situation the DAP needs to choose which data column is appropriate.

2.2.2

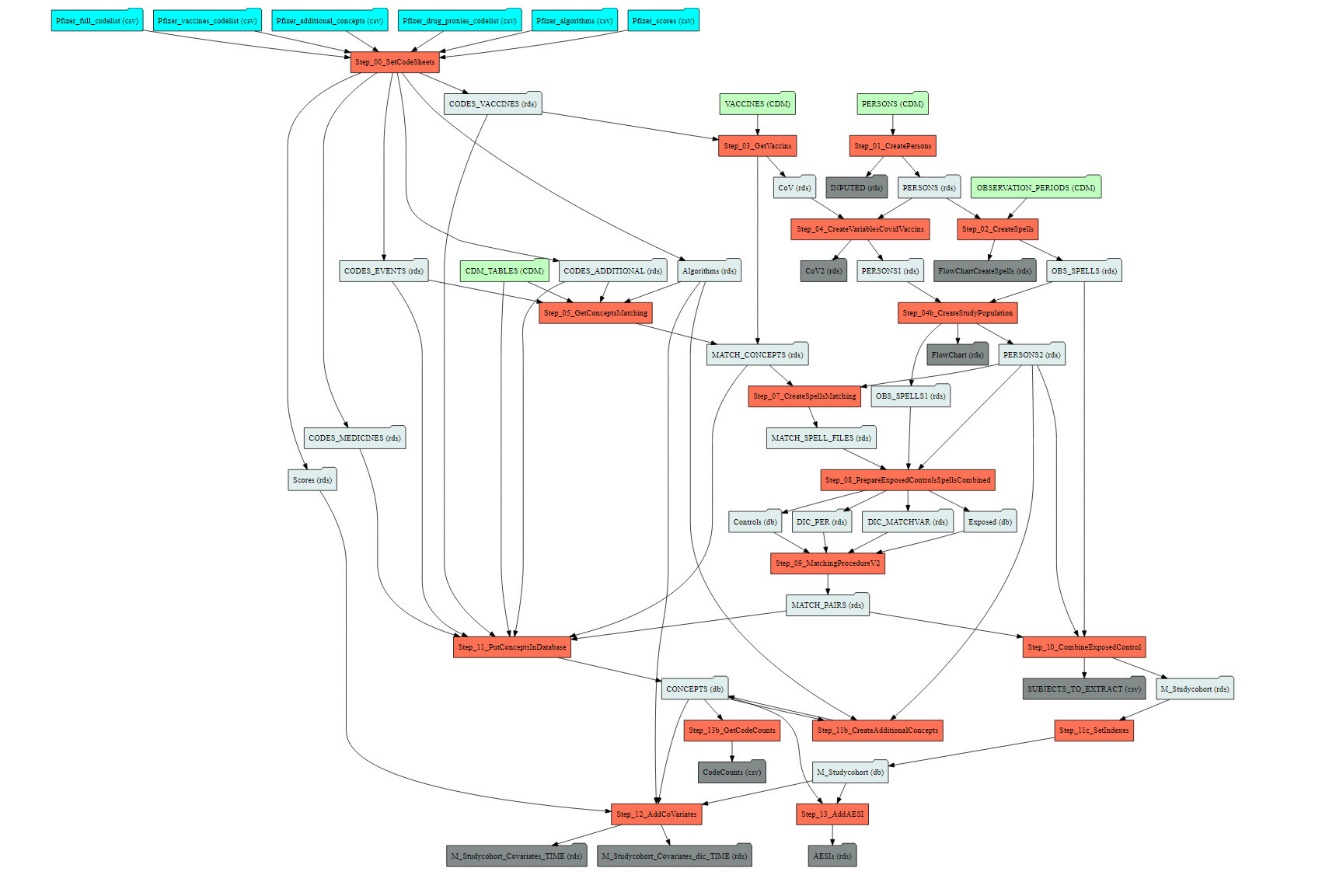
Note that the variable L\_PREGNSTATUS\_COV is not loaded by the script. This is done via a separate script provided by ARS. These results are directly loaded into concepts.db in Step\_05b\_RunPregnancyAlgorithm and is the only concept with a start and an end date.

1. **Information about the script**

**Program.csv**

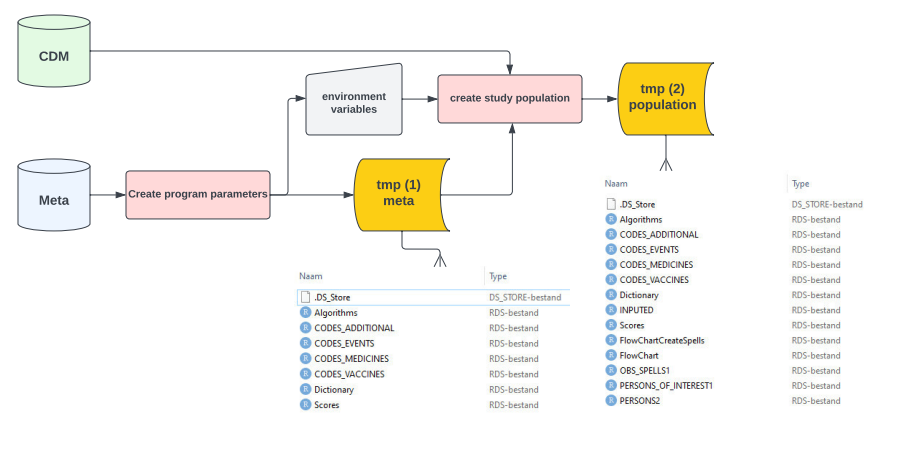
The aim of this file is to document all steps in script. It keeps track of the in/outputs, their formats and their location.

The program.csv is processed in Step\_00\_SetParameters, creating the object PROGRAM that is stored in the global environment. At the first row of every script the function RUN\_SCRIPT stores all in/outputs for that script in an object SCRIPT that is removed at the end of the script. The in/outputs paths are retrieved via the object SCRIPT throughout the script. In this manner the programmer is forced to work via this csv file and the documentation is always up to date and inaccuracies are detected immediately. . TODO: check if on all places in the D3 script this working method is respected (it is additional work so we can discuss if we want this in this way).

****Moreover, based on this file, which should be up to date on all moments in time within the developing process, a codebook and flowchart is generated with the script Step\_99\_CreateMetaData. An example of the flowchart outputted is given below. Since this flowchart is created by a machine based in the program.csv it is very accurate. A more abstract representation of the scrip is given in appendix 1.

Because during the developing process we shifted towards SQLite databases in process that gave problems in working memory, the code to generate the codebook is not accurate anymore. TODO: fix the generation of the codebook.

**Appendix 1: Flow script in combination with meta information**



**Stage 1: preparation**

**Prepare meta information tmp(1)**

**Prepare general population tmp(2)**

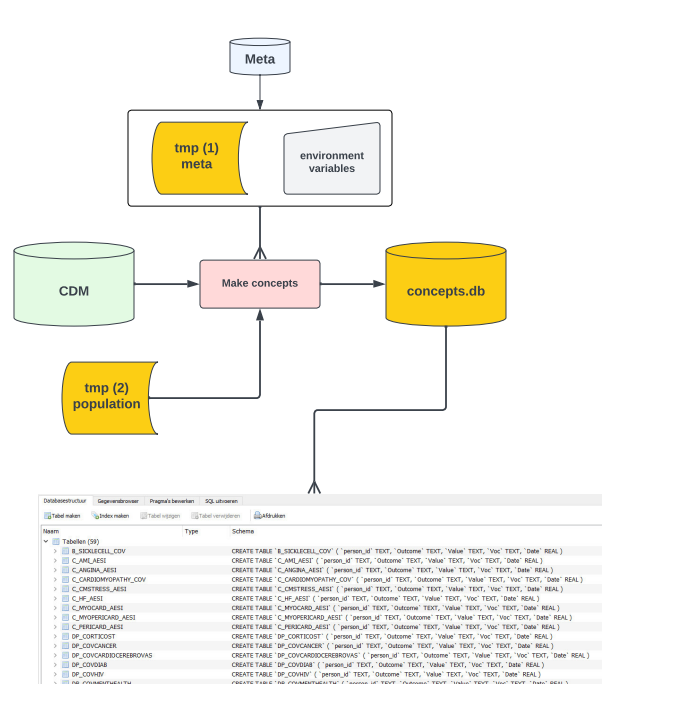
**Scripts:**

Create program parameters

* Step\_00\_SetParameters
* Step\_00\_SetCodeSheets

Create study population

* Step\_01\_CreatePersons
* Step\_02\_CreateSpells
* Step\_03\_GetVaccins (not in figure)
* Step\_04\_CreateVariablesCovidVaccins (not in figure)
* Step\_04b\_CreateStudyPopulation



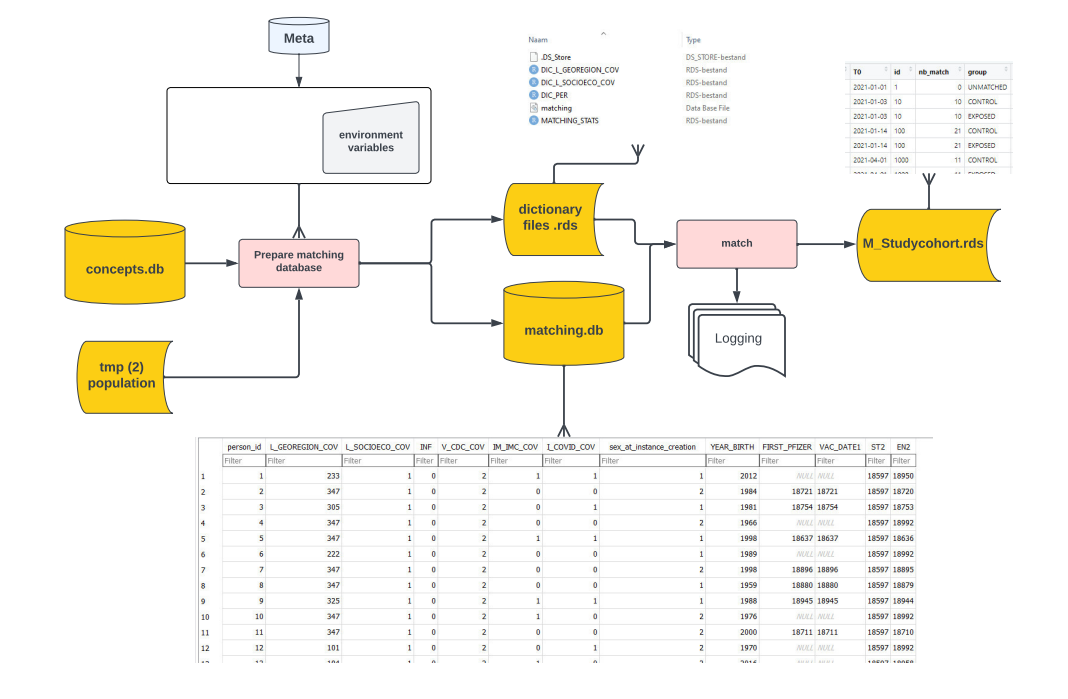
**Stage 2: extracting concepts**

**Only those needed for matching**

**Put those in a database file**

**Scripts:**

* Step\_05\_GetConceptsMatching
* Step\_05b\_RunPregnancyAlgorithm

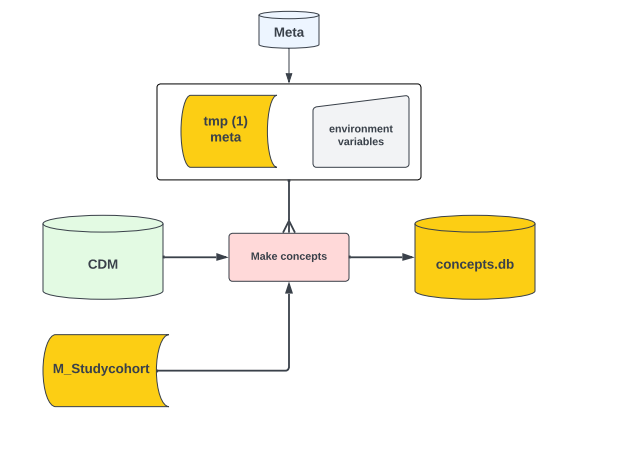


**Stage 3: matching**

**Create match id’s and T0**

**Scripts:**

* Step\_07\_CreateSpellsMatching
* Step\_08\_PrepareExposedControlsSpellsCombined
* Step\_09\_MatchingProcedure
* Step\_10\_CombineExposedControl



**Stage 4: extracting concepts again**

**Only those not needed for matching**

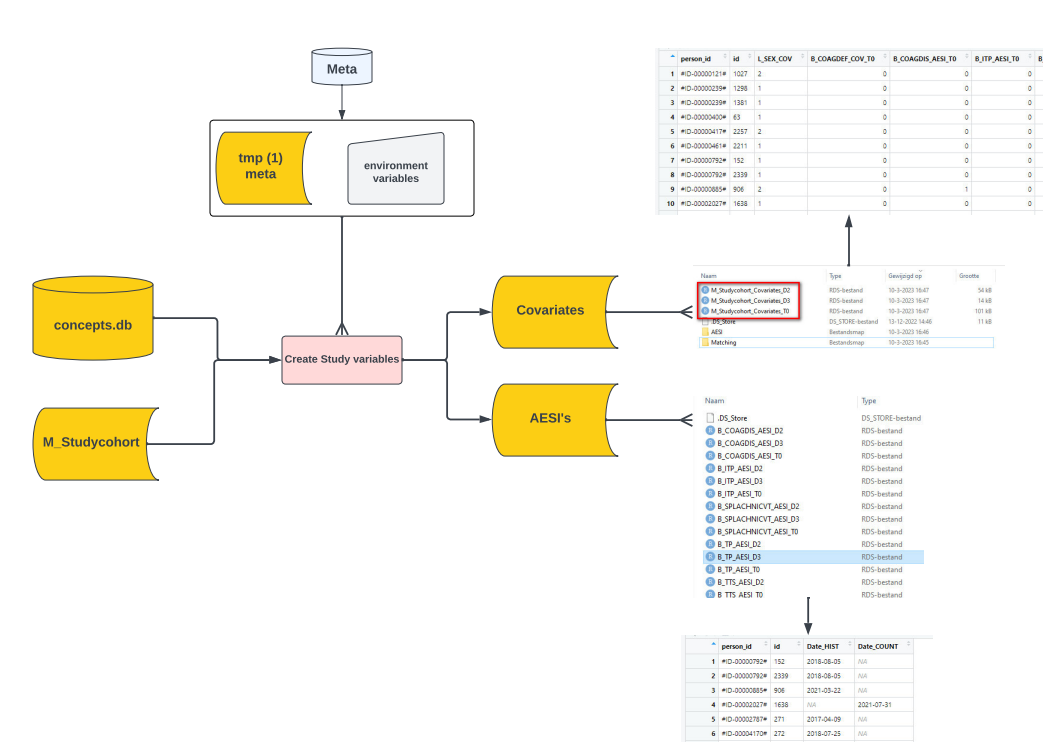
**Only for subjects that are in a match or exposed**

**Add to the existing database file**

**Prepare total database for end game**

**Scripts:**

* Step\_11\_PutConceptsInDatabase
* Step\_11b\_CreateAdditionalConcepts
* Step\_11c\_PrepareDatabase
* Step\_11d\_ApplyDictionaryIntegers
* Step\_11d\_create\_covid\_episodes
* Step\_11e\_create\_covid\_outcomes



**Stage 5: finally creating the desired D3 study variables**

**Relate the concepts to T0 to generate the variables**

**Scripts:**

* Step\_12\_AddCoVariates
* Step\_13\_AddAESI