PharmaSUG China 2019 - Paper AD-047

Metadata-based Auto-Programming Process – Part 2: Map human language to SAS® code by Natural Language Processing (NLP)

Sumesh Kalappurakal, Janssen R&D, New Jersey, US;

Qian Zhao, Harry Chen, Janssen R&D, Shanghai, China;

ABSTRACT

Traditionally the process for programming ADaM datasets is cumbersome and relies heavily on manual work. Per regulatory requirements clinical programing algorithms should be clearly defined in the analysis specification documents in natural language(human-readable). Programmers spend most of the time developing or updating SAS® code according to specification documents. By adopting Machine Learning and leveraging the power of NLP we could analyze human-readable text from the specification documents, train the machine to convert defined algorithms to metadata and map them to the core pieces of SAS® code.

This paper is part 2 of Metadata-based Auto-Programming Process[1], and it shares an approach to automatically generate SAS® code to create ADaM datasets from source SDTM datasets via metadata and NLP methodology.

The strategy would be to extract key information from defined algorithms written in human language and existing code, populate metadata and then utilize the metadata to generate code.

INTRODUCTION

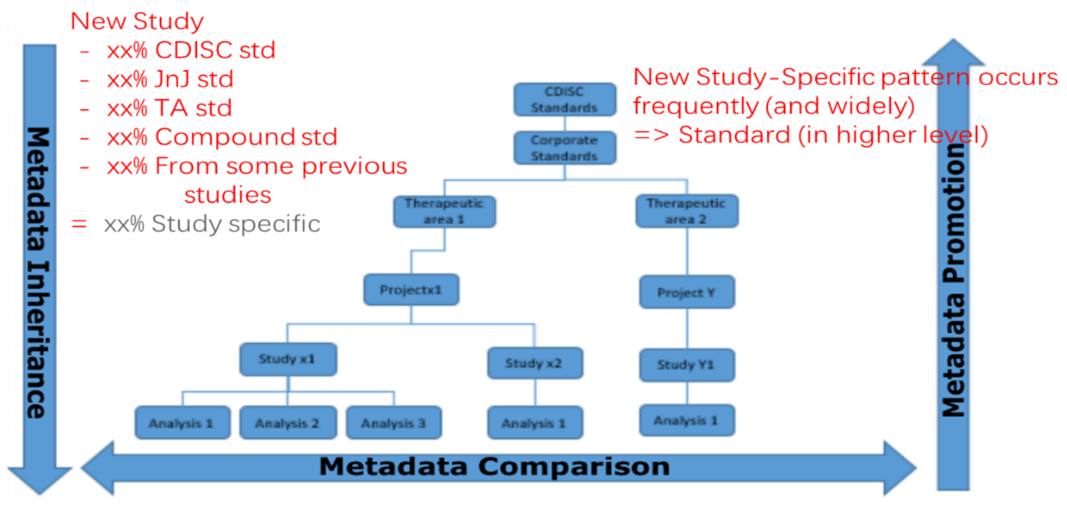
According to Gartner, “by 2020, natural-language generation and artificial intelligence will be a standard feature of 90% of modern business intelligence platforms”[3]. Following the trend, adopting Machine Learning and leveraging the power of NLP we can analyze human language, train the machine to convert defined algorithms to metadata and map them to the core pieces of code (e.g., SAS® or R) to automate the programing process.

We started an innovative project called Autocode in programming team. Autocode project’s main objective is to automate generation of SAS® code for analysis datasets and analysis reports. The vision is to apply new technology to analyze the data structure, digitize the specification document and SAS® code and store them in the form of metadata into a database. Advantages of this model would be more automation, standardization and re-usability study after study. This project is still in the prototype stage and we will share our design, progress and experience in this paper.

FIRST MAIN TOPIC

METADATA DRIVEN APPROACH

With increased enforcement from regulatory agencies towards standardized data and the industry moving towards standardizing analysis requirements, we look to achieve more automation by reusing standard and previously used study algorithms stored in metadata to generate code for analysis.

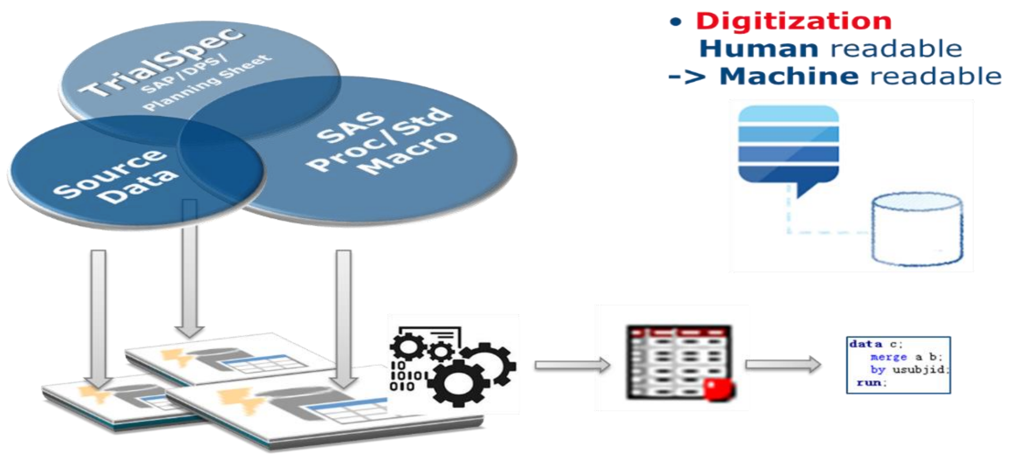


Source: PhUSE 2017 Paper SI02 Clinical Metadata – Metadata Management with a CDISC Mindset

Figure 1. Metadata management

MODULES

Our model is comprised of three different modules to analyze data, SAS® code and document specifications. We use SAS® to calculate the metadata summary for source data and output data. We are using regular expressions and exploring the powerful parser generator ANTLR[10] (Another Tool for Language Recognition) to parse the SAS® code automatically and NLP methodology to analyze specification documents to calculate document similarity and extract key pieces to find the algorithm patterns.



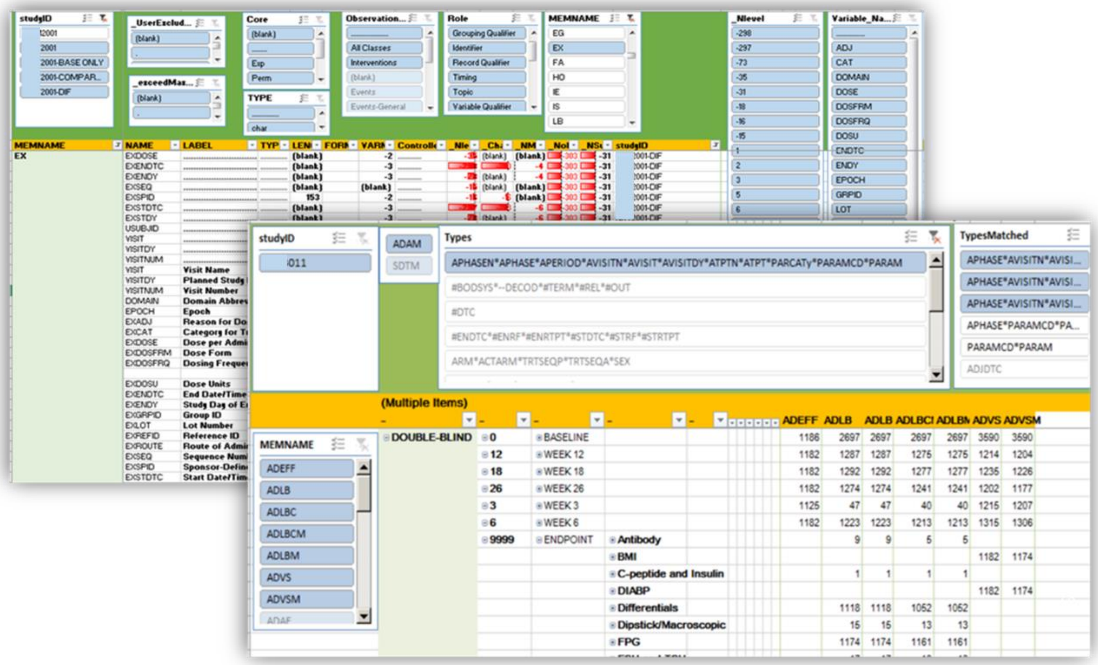
```
data c;
   merge a b;
   by usubjid;
 run;
```

Figure 2. Modules to analyze data, SAS® code and specifications

The goal of the three modules is to extract key information from different sources for data analysis and make them machine readable in structured data, from which we can generate SAS code for analysis. Here are the 3 modules:

1. **PDE (PowerDataExplorer)[2]**: This module is used to generate metadata for SDTM & ADaM datasets, including data elements, structure/hierarchy, relationship, changes/differences across

multiple datasets



studyID: 2001, 2001, 2001-BASE ONLY, 2001-COMPAR..., 2001-DIF

_UserExclud...: (blank), .

_exceedMax...: (blank), .

Core: (blank), Exp, Perm

TYPE: char

Observation...: All Classes, Interventions, (blank), Events, Events-General

Role: Grouping Qualifier, Identifier, Record Qualifier, Timing, Topic, Variable Qualifier

MEMNAME: EG, EX, FA, HO, IE, IS, LB

_Nlevel: -298, -297, -73, -35, -31, -18, -16, -15, 1, 2, 3, 5, 6

Variable_Na...: ADJ, CAT, DOMAIN, DOSE, DOSFRM, DOSFRQ, DOSU, ENDTC, ENDY, EPOCH, GRPID, LOT

MEMNAME	NAME	LABEL	TYP	LEN	FORM	VARN	Controlle	_Nle	_Ch	_NM	_Nol	_NSc	studyID
EX	EXDOSE			(blank)		-2		-35	(blank)	(blank)	-303	-31	2001-DIF
	EXENDTC			(blank)		-3				-4	-303	-31	2001-DIF
	EXENDY			(blank)		-3		-28	(blank)	-4	-303	-31	2001-DIF
	EXSEQ			(blank)		(blank)		-15	(blank)	(blank)	-303	-31	2001-DIF
	EXSPID			153		-2		-15	-5	(blank)	-303	-31	2001-DIF
	EXSTDTC			(blank)		-3				-6	-303	-31	2001-DIF
	EXSTDY			(blank)		-3		-28	(blank)	-6	-303	-31	2001-DIF
	USUBJID												
	VISIT												
	VISITDY												
	VISITNUM												
	VISIT	Visit Name											
	VISITDY	Planned Study											
	VISITNUM	Visit Number											
	DOMAIN	Domain Abbrev											
	EPOCH	Epoch											
	EXADJ	Reason for Do											
	EXCAT	Category for T											
	EXDOSE	Dose per Admi											
	EXDOSFRM	Dose Form											
	EXDOSFRQ	Dosing Frequen											
	EXDOSU	Dose Units											
	EXENDTC	End Date/Time											
	EXENDY	Study Day of E											
	EXGRPID	Group ID											
	EXLOT	Lot Number											
	EXREFID	Reference ID											
	EXROUTE	Route of Admi											
	EXSEQ	Sequence Num											
	EXSPID	Sponsor-Defin											
	EXSTDTC	Start Date/Tim											

studyID: 011

ADAM, SDTM

Types: APHASEN*APHASE*APERIOD*AVISITN*AVISIT*AVISITDY*ATPTN*ATPT*PARCATy*PARAMCD*PARAM, #BODSYS*--DECOD*#TERM*#REL*#OUT, #DTC, #ENDTC*#ENRF*#ENRTPT*#STDTC*#STRF*#STRTPT, ARM*ACTARM*TRTSEQP*TRTSEQA*SEX

TypesMatched: APHASE*AVISITN*AVISI..., APHASE*AVISITN*AVISI..., APHASE*AVISITN*AVISI..., APHASE*PARAMCD*PA..., PARAMCD*PARAM, ADJDTC

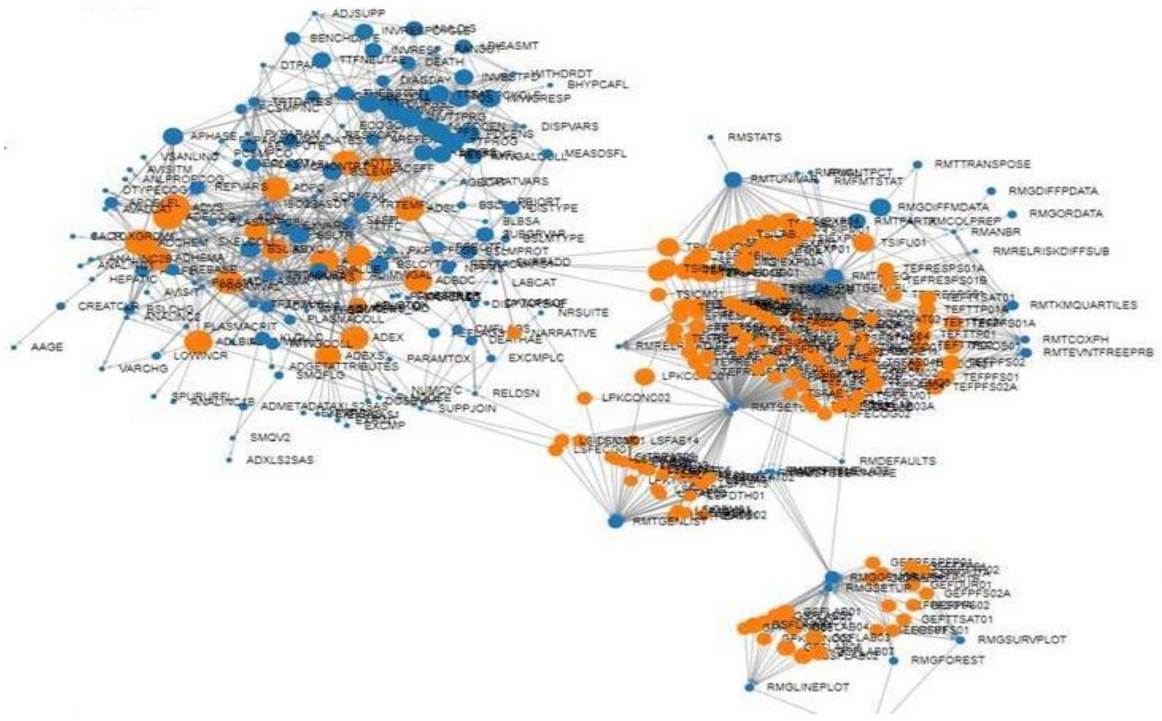
MEMNAME: ADEFF, ADLB, ADLBC, ADLBCM, ADLBM, ADVS, ADVSM, ADAE

(Multiple Items)

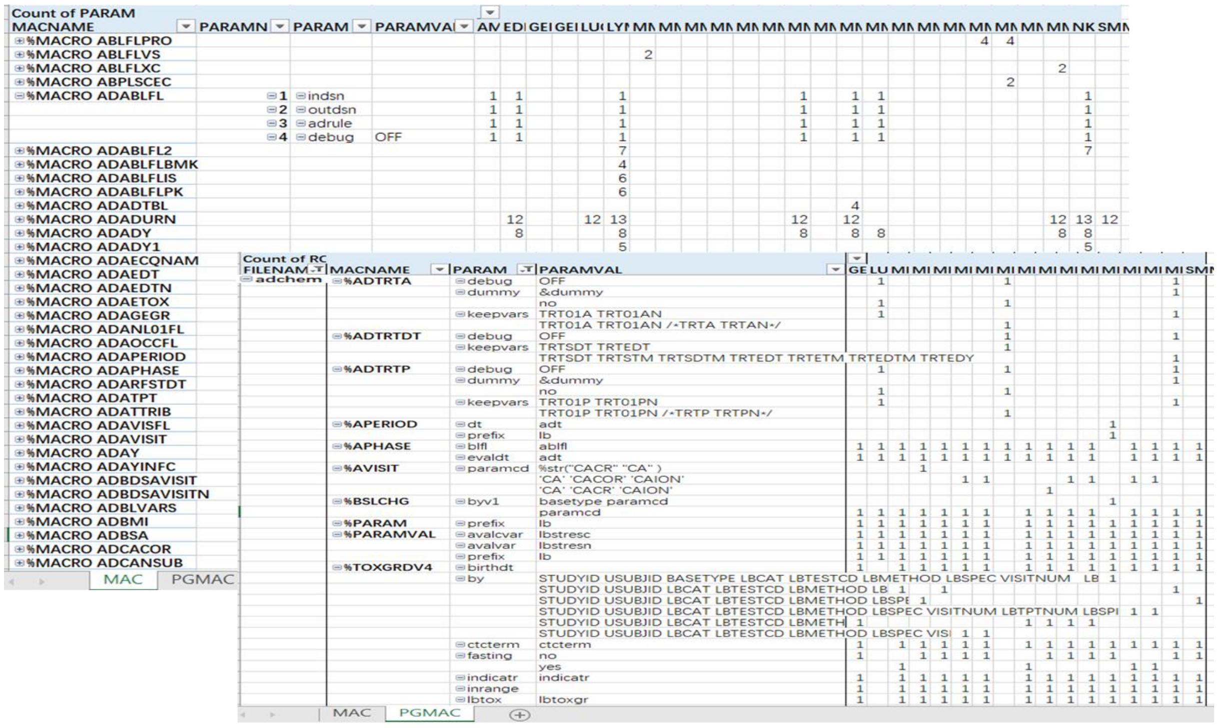
-	-	-	-	-	ADEFF	ADLB	ADLB	ADLBCI	ADLBM	ADVS	ADVSM
DOUBLE-BLIND	0	BASELINE			1186	2697	2697	2697	2697	3590	3590
	12	WEEK 12			1182	1287	1287	1275	1275	1214	1204
	18	WEEK 18			1182	1292	1292	1277	1277	1235	1226
	26	WEEK 26			1182	1274	1274	1241	1241	1202	1177
	3	WEEK 3			1125	47	47	40	40	1215	1207
	6	WEEK 6			1182	1223	1223	1213	1213	1315	1306
	9999	ENDPOINT	Antibody			9	9	5	5		
			BMI							1182	1174
			C-peptide and Insulin			1	1	1	1		
			DIABP							1182	1174
			Differentials			1118	1118	1052	1052		
			Dipstick/Macroscopic			15	15	13	13		
			FPG			1174	1174	1161	1161		

Figure 3. PDE Module

2. **PME (PowerMacroExplorer):** This module is used to create a dataset which contains program/macro dependency, macro parameters and macro calls in historical use cases by scanning and parsing codes across multiple study folders



3



Count of PARAM

MACNAME	PARAMN	PARAM	PARAMVAL	AN	ED	GEI	GEI	LU(	LYI	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	MN	NK	SMN
⊞%MACRO ABLFLPRO																							4	4				
⊞%MACRO ABLFLVS										2																		
⊞%MACRO ABLFLXC																										2		
⊞%MACRO ABPLSCEC																								2				
⊟%MACRO ADABLFL	⊟1	⊟indsn		1	1				1							1		1	1								1	
	⊟2	⊟outdsn		1	1				1							1		1	1								1	
	⊟3	⊟adrule		1	1				1							1		1	1								1	
	⊟4	⊟debug	OFF	1	1				1							1		1	1								1	
⊞%MACRO ADABLFL2									7																		7	
⊞%MACRO ADABLFLBMK									4																			
⊞%MACRO ADABLFLIS									6																			
⊞%MACRO ADABLFLPK									6																			
⊞%MACRO ADADTBL																		4										
⊞%MACRO ADADURN					12			12	13							12		12								12	13	12
⊞%MACRO ADADY					8				8							8		8	8							8	8	
⊞%MACRO ADADY1									5																		5	
⊞%MACRO ADAECQNAM																												
⊞%MACRO ADAEDT																												
⊞%MACRO ADAEDTN																												
⊞%MACRO ADAETOX																												
⊞%MACRO ADAGEGR																												
⊞%MACRO ADANL01FL																												
⊞%MACRO ADAOCCFL																												
⊞%MACRO ADAPERIOD																												
⊞%MACRO ADAPHASE																												
⊞%MACRO ADARFSTDT																												
⊞%MACRO ADATPT																												
⊞%MACRO ADATTRIB																												
⊞%MACRO ADAVISFL																												
⊞%MACRO ADAVISIT																												
⊞%MACRO ADAY																												
⊞%MACRO ADAYINFC																												
⊞%MACRO ADBDSAVISIT																												
⊞%MACRO ADBDSAVISITN																												
⊞%MACRO ADBLVARS																												
⊞%MACRO ADBMI																												
⊞%MACRO ADBSA																												
⊞%MACRO ADCACOR																												
⊞%MACRO ADCANSUB																												

MAC PGMAC

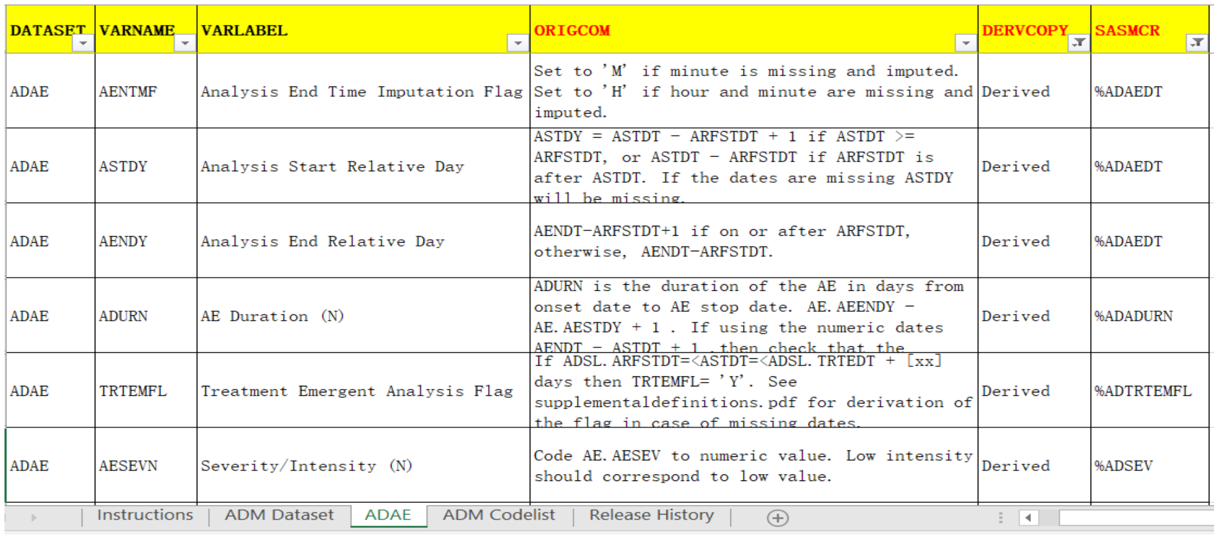
Count of RC

FILENAM	MACNAME	PARAM	PARAMVAL	GE	LU	MI	MI	MI	MI	MI	MI	MI	MI	MI	MI	MI	MI	MI	MI	SMN
⊟adchem	⊟%ADTRTA	⊟debug	OFF		1						1								1	
		⊟dummy	&dummy																1	
			no		1						1									
		⊟keepvars	TRT01A TRT01AN		1														1	
			TRT01A TRT01AN /*TRTA TRTAN*/								1									
	⊟%ADTRTDT	⊟debug	OFF								1								1	
		⊟keepvars	TRTSDT TRTEDT								1									
			TRTSDT TRTSTM TRTSDTM TRTEDT TRTETM TRTEDTM TRTEDY																1	
	⊟%ADTRTP	⊟debug	OFF		1						1								1	
		⊟dummy	&dummy																1	
			no		1						1									
		⊟keepvars	TRT01P TRT01PN		1														1	
			TRT01P TRT01PN /*TRTP TRTPN*/								1									
	⊟%APERIOD	⊟dt	adt													1				
		⊟prefix	lb													1				
	⊟%APHASE	⊟blfl	ablfl	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1
		⊟evaldt	adt	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1
	⊟%AVISIT	⊟paramcd	%str("CACR" "CA")				1													
			'CA' 'CACOR' 'CAION'						1	1				1	1		1	1		
			'CA' 'CACR' 'CAION'										1							
	⊟%BSLCHG	⊟byv1	basetype paramcd													1				
			paramcd	1	1	1	1	1	1	1		1	1	1	1		1	1	1	1
	⊟%PARAM	⊟prefix	lb	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1
	⊟%PARAMVAL	⊟avalcvar	lbstresc	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟avalvar	lbstresn	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟prefix	lb	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1
	⊟%TOXGRDV4	⊟birthdt		1		1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟by	STUDYID USUBJID BASETYPE LBCAT LBTESTCD LBMETHOD LBSPEC VISITNUM LB													1				
			STUDYID USUBJID LBCAT LBTESTCD LBMETHOD LB			1		1											1	
			STUDYID USUBJID LBCAT LBTESTCD LBMETHOD LBSPE				1													1
			STUDYID USUBJID LBCAT LBTESTCD LBMETHOD LBSPEC VISITNUM LBTPTNUM LBSPI														1	1		
			STUDYID USUBJID LBCAT LBTESTCD LBMETH	1								1	1	1	1					
			STUDYID USUBJID LBCAT LBTESTCD LBMETHOD LBSPEC VISI						1	1										
		⊟ctcterm	ctcterm	1		1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟fasting	no	1			1	1	1	1			1	1	1	1			1	1
			yes			1						1					1	1		
		⊟indicatr	indicatr	1		1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟inrange		1		1	1	1	1	1		1	1	1	1	1	1	1	1	1
		⊟lbtox	lbtoxgr	1		1	1	1	1	1		1	1	1	1	1	1	1	1	1

MAC PGMAC

Figure 4. PME Module

3. **ADaM Planning Sheet analyzed by NLP:** The Excel specification includes key information such as SAS macro name (SASMCR) and text descriptions of the derivation algorithm (ORIGCOM) necessary to guide programmer to create SAS code for the variable derivations. Generally, ORIGCOM columns may be any combination of plain English text, a formula, pseudocode, etc. It describes the source of the data or the derivation clearly and it serves as the submission-ready text for the source/derivation/comments in define.xml. This paper focus on derivation and we will introduce how to analyze the algorithm text in ORIGCOM using NLP to map it to macro metadata and generate SAS code. At this stage we focus on the planning sheet, but a similar approach can be applied to other document specifications like protocol and Statistical Analysis Plan (SAP).



DATASET	VARNAME	VARLABEL	ORIGCOM	DERVCOPY	SASMCR
ADAE	AENTMF	Analysis End Time Imputation Flag	Set to 'M' if minute is missing and imputed. Set to 'H' if hour and minute are missing and imputed.	Derived	%ADAEDT
ADAE	ASTDY	Analysis Start Relative Day	ASTDY = ASTDT - ARFSTDT + 1 if ASTDT >= ARFSTDT, or ASTDT - ARFSTDT if ARFSTDT is after ASTDT. If the dates are missing ASTDY will be missing.	Derived	%ADAEDT
ADAE	AENDY	Analysis End Relative Day	AENDT-ARFSTDT+1 if on or after ARFSTDT, otherwise, AENDT-ARFSTDT.	Derived	%ADAEDT
ADAE	ADURN	AE Duration (N)	ADURN is the duration of the AE in days from onset date to AE stop date. AE.AEENDY - AE.AESTDY + 1 . If using the numeric dates AENDT - ASTDT + 1 , then check that the	Derived	%ADADURN
ADAE	TRTEMFL	Treatment Emergent Analysis Flag	If ADSL.ARFSTDT=<ASTDT=<ADSL.TRTEDT + [xx] days then TRTEMFL='Y'. See supplementaldefinitions.pdf for derivation of the flag in case of missing dates.	Derived	%ADTRTEMFL
ADAE	AESEVN	Severity/Intensity (N)	Code AE.AESEV to numeric value. Low intensity should correspond to low value.	Derived	%ADSEV

Instructions | ADM Dataset | ADAE | ADM Codelist | Release History

Figure 5. ADaM Planning Sheet example

Natural Language Processing (NLP)

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding[4]. NLP is important because it helps analyze large volumes of textual data efficiently and resolve ambiguity in language to structure a highly unstructured textual data source. Below is a simple example of syntax analysis to discover structure of the algorithm text.

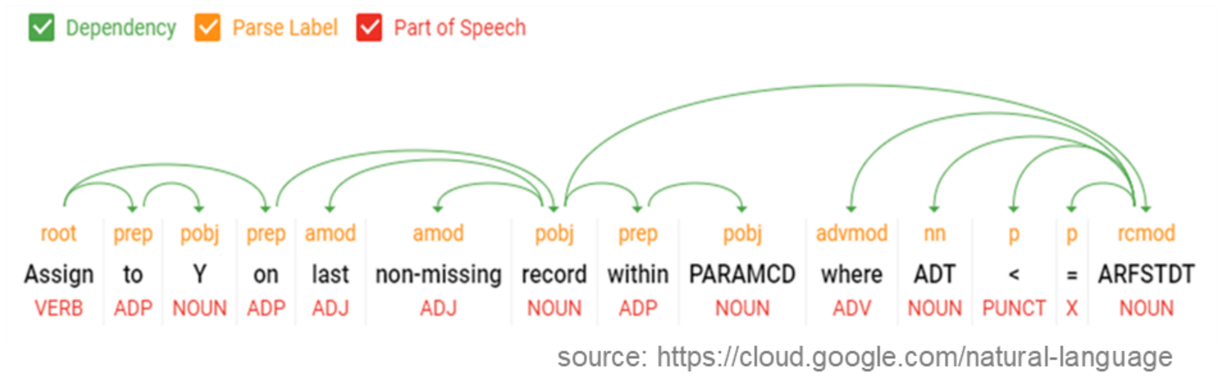


Figure 6. Syntax analysis from NLP

We have loaded 600,000+ records of variable derivation algorithms into our prototype Metadata Repository (MDR) system collected from the planning sheets across 60+ compounds and 1000+ study folders. Taking the variable ABLFL (Baseline Record Flag) derivation for example, there are 3000+ records of algorithms and 380 unique algorithm descriptions. After some pre-processing steps to normalize (lemmatization and removing some stop words) the texts, we did basic syntax analysis to get some idea of the key nouns, adjectives and their relationships in ABLFL derivation, like the words ‘first’, ‘last’, ‘closest’, ’multiple’, ’average’ etc. that play important role to describe the algorithms to select the right value from data.

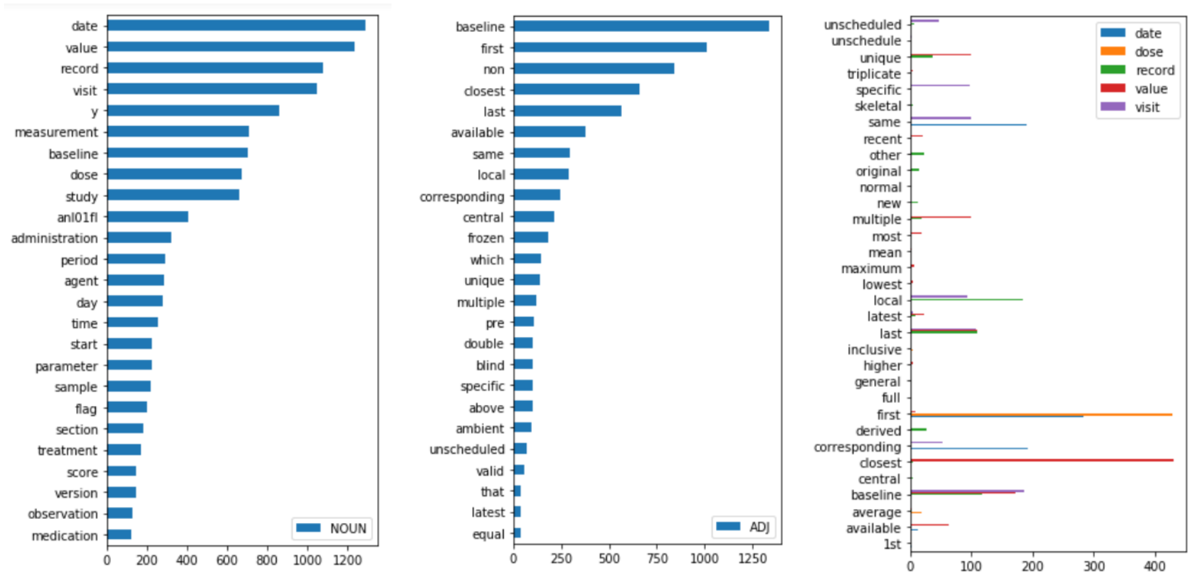


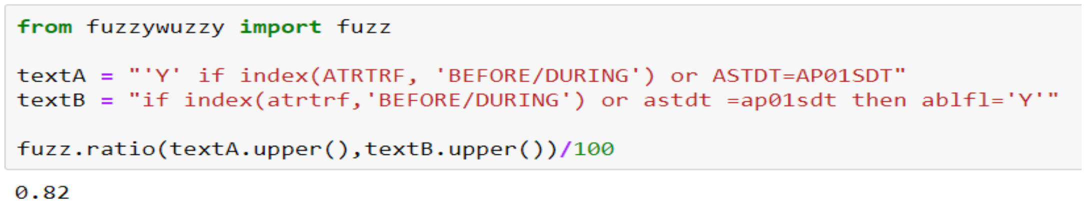
Figure 7. Syntax analysis for variable ABLFL derivation

There are two approaches developed to map the human-language defined algorithms to code. The unsupervised approach is to calculate the document similarity of algorithm sentences to get the corresponding standard/historical codes. The supervised approach is to split algorithm sentences to phrases and map phrases to code pieces in a training database and then incorporate the RNN (Recurrent Neural Networks)[11] model to learn the pattern and generate more flexible code.

.

- **Unsupervised approach:** For new data analysis algorithms we search the database based on text similarity score to get the most similar standard/historical algorithms and codes. We applied two approaches to calculate the text similarity, Fuzzy string-matching algorithm based on edit distance, and Word Mover's Distance (WMD) [12] algorithm based on semantic distance, and we also created the algorithm clusters from the similarity matrix.

    1. Fuzzy String Matching, also called Approximate String Matching, is the process of finding strings that approximatively match a given pattern. The degree of closeness between two strings is measured using Levenshtein Distance[14], also known as edit distance which basically is based on counting the number of primitive operations (insertion, deletion, and substitution) necessary to convert the string into an exact match. We use the fuzzywuzzy[6] package to do the calculation:



```
from fuzzywuzzy import fuzz

textA = "'Y' if index(ATRTRF, 'BEFORE/DURING') or ASTDT=AP01SDT"
textB = "if index(atrtrf,'BEFORE/DURING') or astdt =ap01sdt then ablfl='Y'"

fuzz.ratio(textA.upper(),textB.upper())/100
```

```
0.82
```

Figure 8. Text similarity score using fuzzywuzzy

    2. Word Mover's Distance (WMD) algorithm is a method that allows us to assess the "distance" between two documents in a meaningful way, even when they have no words in common, via vector embeddings of words. Word embeddings (word vectors) are numeric representations of words, usually generated via dimensionality reduction on a word cooccurrence matrix for a large corpus. These vectors are used to calculate semantic similarity between words and documents:

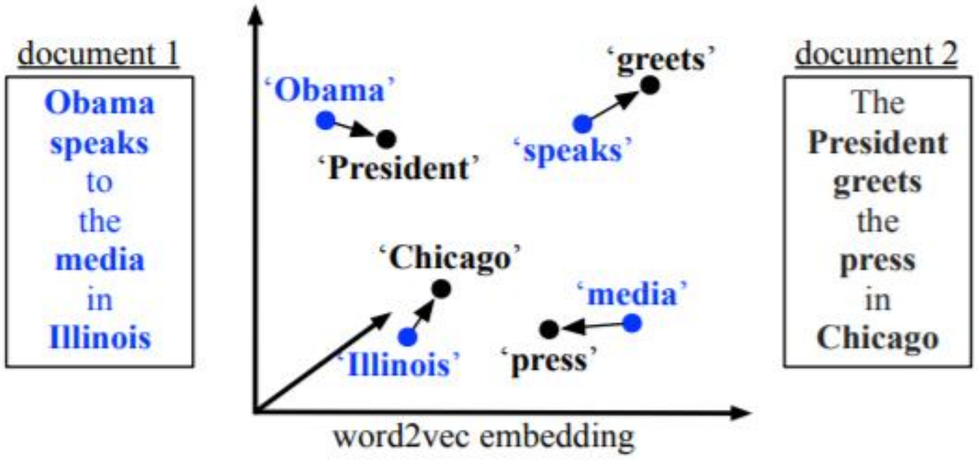
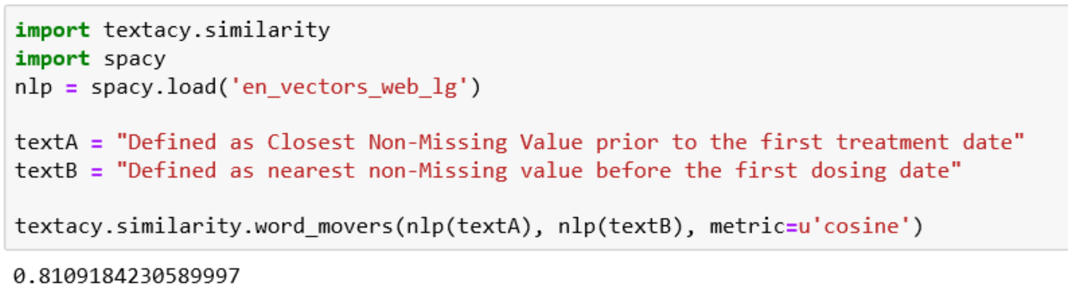


Figure 9. WMD algorithm via word embedding

We use the library textacy[7] which is built on top of spaCy[8] package to compute the word mover's distance and get the text similarity score



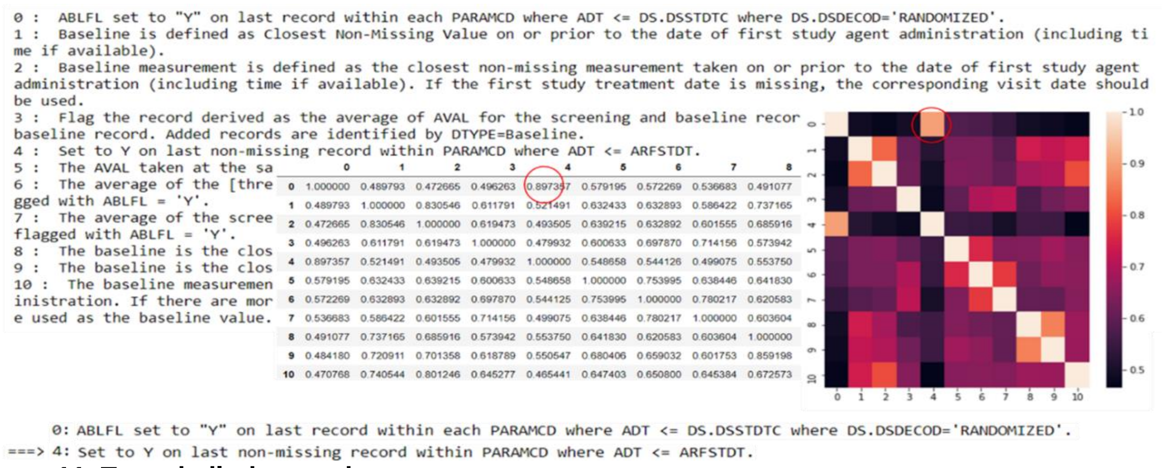
```python
import textacy.similarity
import spacy
nlp = spacy.load('en_vectors_web_lg')

textA = "Defined as Closest Non-Missing Value prior to the first treatment date"
textB = "Defined as nearest non-Missing value before the first dosing date"

textacy.similarity.word_movers(nlp(textA), nlp(textB), metric=u'cosine')
```

```
0.8109184230589997
```

Figure 10. Text semantic similarity score using WMD

We loaded pre-trained 'en_vectors_web_lg'[13] model which provides 300-dimensional vectors for over 1 million terms of English, and we can also create our own word embeddings to cover more domain specific vocabularies in our industry if we have more training data.

3. For a group of algorithms, we calculate similarity score for each pair of algorithm texts and generate the similarity matrix, based on which we are able to find the most similar pairs (or top 3 most similar ones), also hierarchical clustering dendrogram plots are created to show algorithm clusters.



```
0 :  ABLFL set to "Y" on last record within each PARAMCD where ADT <= DS.DSSTDTC where DS.DSDECOD='RANDOMIZED'.
1 :  Baseline is defined as Closest Non-Missing Value on or prior to the date of first study agent administration (including ti
me if available).
2 :  Baseline measurement is defined as the closest non-missing measurement taken on or prior to the date of first study agent
administration (including time if available). If the first study treatment date is missing, the corresponding visit date should
be used.
3 :  Flag the record derived as the average of AVAL for the screening and baseline recor
baseline record. Added records are identified by DTYPE=Baseline.
4 :  Set to Y on last non-missing record within PARAMCD where ADT <= ARFSTDT.
5 :  The AVAL taken at the sa
6 :  The average of the [thre
gged with ABLFL = 'Y'.
7 :  The average of the scree
flagged with ABLFL = 'Y'.
8 :  The baseline is the clos
9 :  The baseline is the clos
10 :  The baseline measuremen
inistration. If there are mor
e used as the baseline value.
```

	0	1	2	3	4	5	6	7	8
0	1.000000	0.489793	0.472665	0.496263	0.897357	0.579195	0.572269	0.536683	0.491077
1	0.489793	1.000000	0.830546	0.611791	0.521491	0.632433	0.632893	0.586422	0.737165
2	0.472665	0.830546	1.000000	0.619473	0.493505	0.639215	0.632892	0.601555	0.685916
3	0.496263	0.611791	0.619473	1.000000	0.479932	0.600633	0.697870	0.714156	0.573942
4	0.897357	0.521491	0.493505	0.479932	1.000000	0.548658	0.544126	0.499075	0.553750
5	0.579195	0.632433	0.639215	0.600633	0.548658	1.000000	0.753995	0.638446	0.641830
6	0.572269	0.632893	0.632892	0.697870	0.544125	0.753995	1.000000	0.780217	0.620583
7	0.536683	0.586422	0.601555	0.714156	0.499075	0.638446	0.780217	1.000000	0.603604
8	0.491077	0.737165	0.685916	0.573942	0.553750	0.641830	0.620583	0.603604	1.000000
9	0.484180	0.720911	0.701358	0.618789	0.550547	0.680406	0.659032	0.601753	0.859198
10	0.470768	0.740544	0.801246	0.645277	0.465441	0.647403	0.650800	0.645384	0.672573

```
    0: ABLFL set to "Y" on last record within each PARAMCD where ADT <= DS.DSSTDTC where DS.DSDECOD='RANDOMIZED'.
===> 4: Set to Y on last non-missing record within PARAMCD where ADT <= ARFSTDT.
```

Figure 11. Text similarity matrix

We have noticed that the algorithm descriptions in new studies are becoming more and more standardized than that in the legacy studies, as shown below, which lays great foundation to more automation over time.

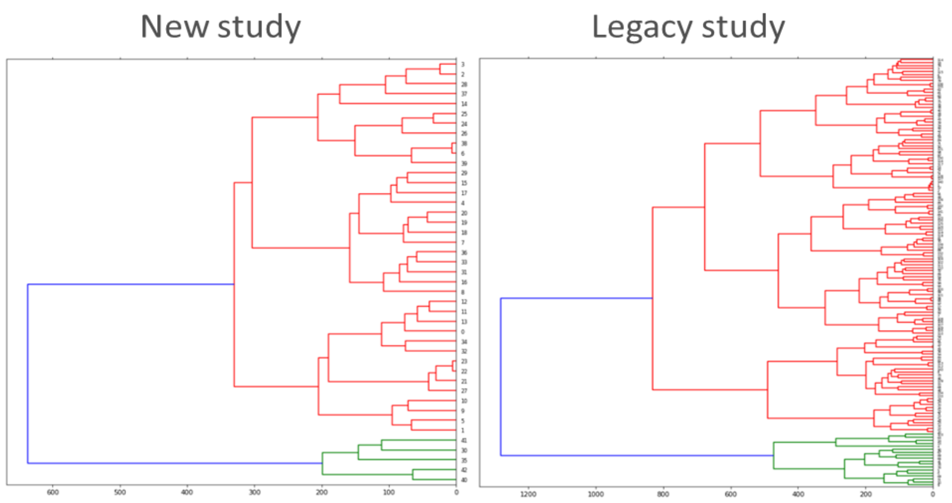
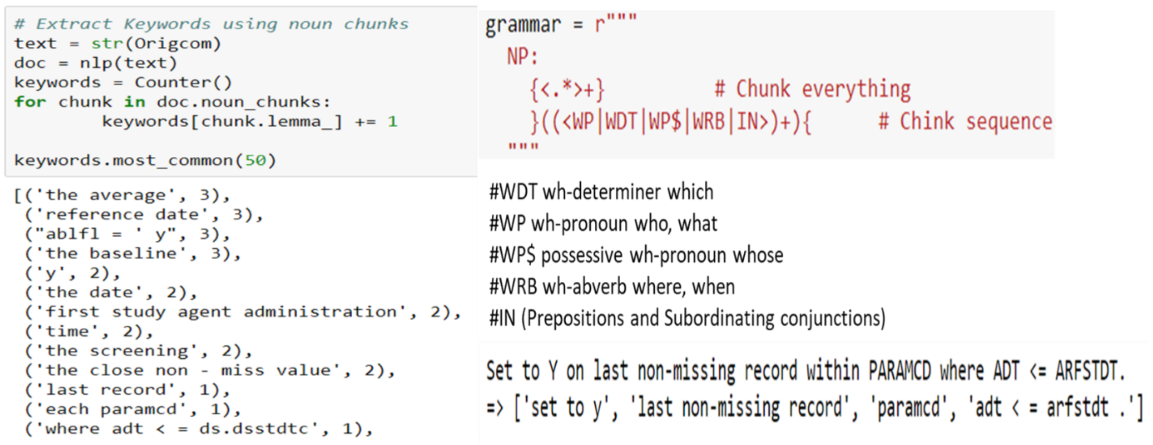


Figure 12. Comparison between new study and legacy study

- **Supervised approach:** Small pieces of algorithms are much easier to be matched and reused than long sentences and we can map the algorithm to code based on pieces. The NLP software NLTK[9] and spaCy are used to split the algorithm documentation to meaningful pieces, and the powerful parser generator ANTLR is used to parse the SAS® code (both macro calls and data steps), and then the language pieces and code pieces would be mapped and combined to build the training database prepared for the RNN (Recurrent Neural Networks) deep learning model, based on which machine can learn the algorithm pattern and so map the new algorithm to code automatically and more flexibly. Below are the 5 steps and steps 3-5 are in a proof of concept stage.

  1. Split algorithm texts into pieces using NLP tools (NLTK and spaCy) with manually reviews:



```
# Extract Keywords using noun chunks
text = str(Origcom)
doc = nlp(text)
keywords = Counter()
for chunk in doc.noun_chunks:
            keywords[chunk.lemma_] += 1

keywords.most_common(50)
```

```
[('the average', 3),
 ('reference date', 3),
 ("ablfl = ' y", 3),
 ('the baseline', 3),
 ('y', 2),
 ('the date', 2),
 ('first study agent administration', 2),
 ('time', 2),
 ('the screening', 2),
 ('the close non - miss value', 2),
 ('last record', 1),
 ('each paramcd', 1),
 ('where adt < = ds.dsstdtc', 1),
```

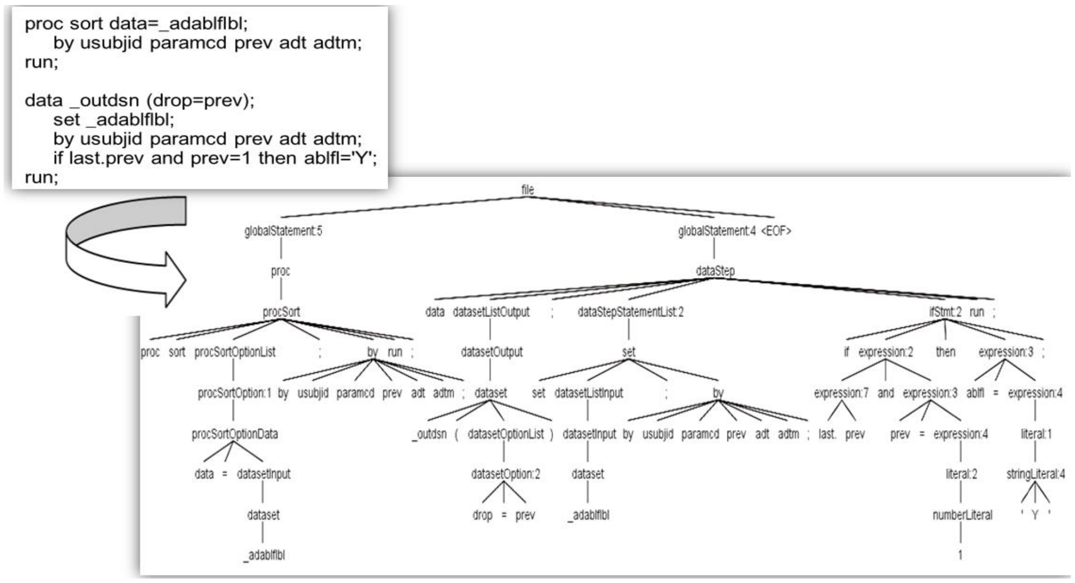
```
grammar = r"""
  NP:
    {<.*>+}          # Chunk everything
    }((<WP|WDT|WP$|WRB|IN>)+){       # Chink sequence
"""
```

```
#WDT wh-determiner which
#WP wh-pronoun who, what
#WP$ possessive wh-pronoun whose
#WRB wh-abverb where, when
#IN (Prepositions and Subordinating conjunctions)
```

```
Set to Y on last non-missing record within PARAMCD where ADT <= ARFSTDT.
=> ['set to y', 'last non-missing record', 'paramcd', 'adt < = arfstdt .']
```

Display 1. Split texts into pieces with NLTK and spaCy

  2. Parse codes using ANTLR and extract key parameters:



```
proc sort data=_adablflbl;
    by usubjid paramcd prev adt adtm;
run;

data _outdsn (drop=prev);
    set _adablflbl;
    by usubjid paramcd prev adt adtm;
    if last.prev and prev=1 then ablfl='Y';
run;
```

file
globalStatement:5
globalStatement:4 <EOF>
proc
dataStep
procSort
data datasetListOutput ; dataStepStatementList:2
ifStmt:2 run ;
proc sort procSortOptionList ; by run ;
datasetOutput
set
if expression:2 then expression:3 ;
procSortOption:1 by usubjid paramcd prev adt adtm ; dataset
set datasetListInput ; by
expression:7 and expression:3 ablfl = expression:4
procSortOptionData
_outdsn (datasetOptionList)
datasetInput by usubjid paramcd prev adt adtm ;
last. prev
prev = expression:4
literal:1
data = datasetInput
datasetOption:2
dataset
literal:2
stringLiteral:4
dataset
drop = prev
_adablflbl
numberLiteral
' Y '
_adablflbl
1

Display 2. Code parsing with ANTLR

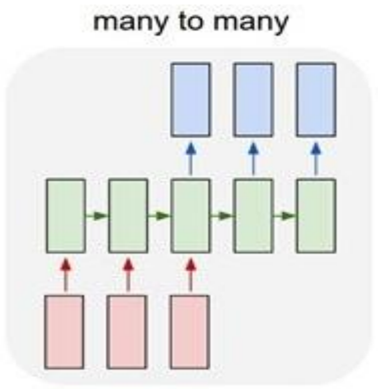
3. Map algorithm text pieces to macro name and parameters in the training data

Assign to Y on last non-missing record within PARAMCD where ADT <= ARFSTDT.

%Assign(indsn=xxx ,value='Y' ,by=PARAMCD ,select=last non-missing ,subset=%str(.<ADT<=ARFSTDT))

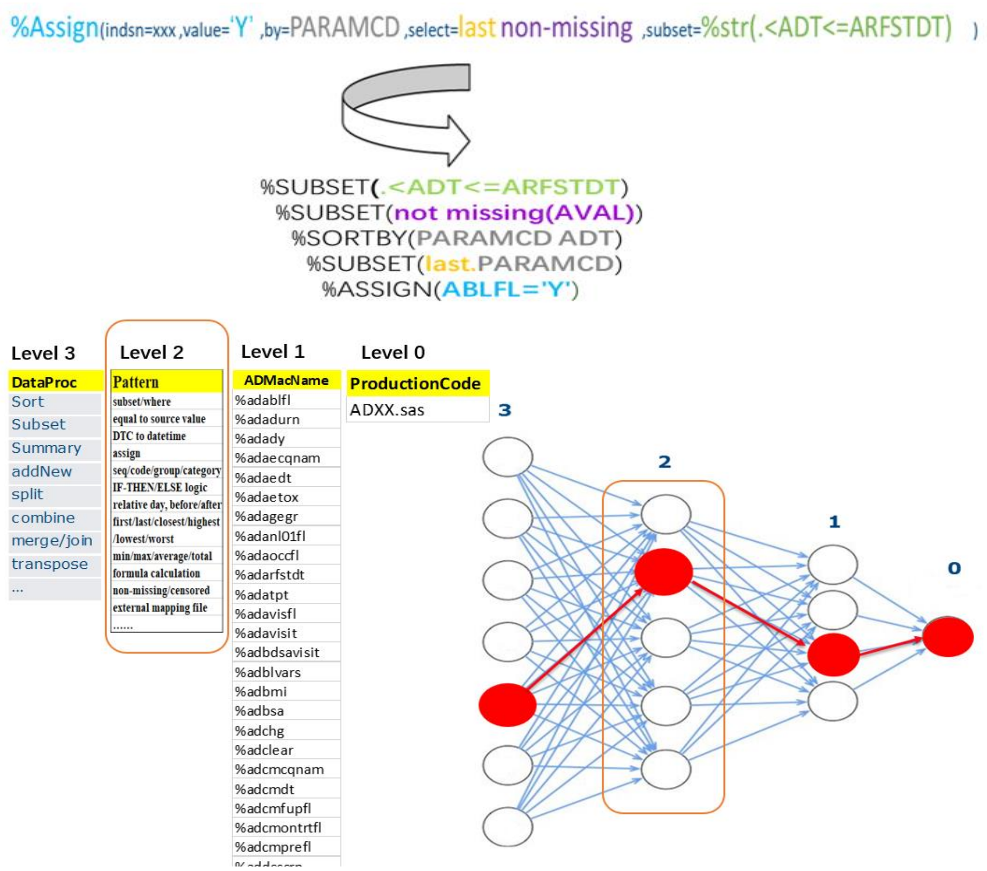
Display 3. Map into the macro name and parameters in the training data

4. Train an RNN model



Display 4. Training of RNN model

5. Complex algorithm decomposed into pattern algorithms



```
%Assign(indsn=xxx,value='Y',by=PARAMCD,select=last non-missing,subset=%str(.<ADT<=ARFSTDT) )
```

```
%SUBSET(.<ADT<=ARFSTDT)
 %SUBSET(not missing(AVAL))
  %SORTBY(PARAMCD ADT)
   %SUBSET(last.PARAMCD)
    %ASSIGN(ABLFL='Y')
```

Level 3	Level 2	Level 1	Level 0
DataProc	Pattern	ADMacName	ProductionCode
Sort	subset/where	%adablfl	ADXX.sas
Subset	equal to source value	%adadurn	
Summary	DTC to datetime	%adady	
addNew	assign	%adaecqnam	
split	seq/code/group/category	%adaedt	
combine	IF-THEN/ELSE logic	%adaetox	
merge/join	relative day, before/after	%adagegr	
transpose	first/last/closest/highest	%adanl01fl	
...	/lowest/worst	%adaoccfl	
	min/max/average/total	%adarfstdt	
	formula calculation	%adatpt	
	non-missing/censored	%adavisfl	
	external mapping file	%adavisit	
	......	%adbdsavisit	
		%adblvars	
		%adbmi	
		%adbsa	
		%adchg	
		%adclear	
		%adcmcqnam	
		%adcmdt	
		%adcmfupfl	
		%adcmontrtfl	
		%adcmprefl	

Display 5. Decompose complex algorithm into pattern algorithm

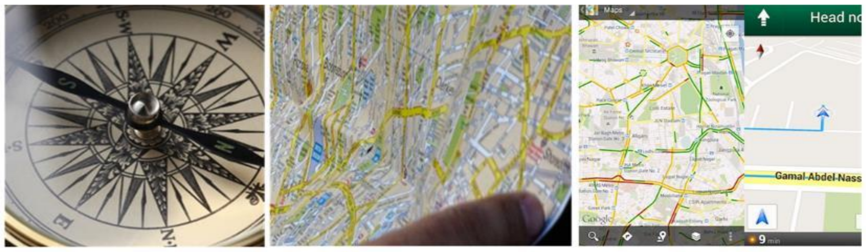
For the deep learning model, we need to prepare large amount of training data (language pieces with corresponding code pieces). This is challenging because we need high-quality data, and this involves lots of texts and codes preprocessing and manual review, compared to the unsupervised learning process.

CONCLUSION

We have designed and explored Metadata-based Auto-Programming Process with NLP to convert three different parts (data, SAS® code and document specifications) into machine readable metadata and connect them to try to achieve as much automation as possible in code generation for data analysis. From our experience, the analysis standard, high quality training data and technology are key factors for the success. Technological progress follows a pattern of exponential growth and there will be more and more powerful Machine Learning and NLP technology available to our daily work over time, so the top priority for us is to focus on the business side and collect 'big' standardized data, or Machine Learning cannot help as "garbage in, garbage out".

The benefits of Metadata-based Auto-Programming Process are:

- Top-Down Standard end-to-end metadata management and project management
- Metadata-driven SAS code generation to reduce manual work and potential human error which inevitably leads to wasted time in both coding and debugging
- Significant reduction in time to submission and increase in consistency and quality
- Accumulate and maintain internal knowledge in database and system
- Leave programmers more time for more complex tasks, in depth disease area understanding, more time for broadening skillsets



Exploration / Map / Auto-Navigation
Open-Code / Standard/ Auto-Code
Experience / Documentation/ System automation

In a fast-changing world, technology is changing how we live. Like Auto-Navigation brings convenience in our daily life, compared to the map and compass age. Humans needs keep-up with the fast pace technology advancement.

We are only just scratching the surface when it comes to the uses of AI and Machine Learning in the Pharmaceutical industry. However, even at this early stage, the technologies are proving to be tremendously promising when it comes to giving new mechanistic insights to disease and thereby helping to identify promising targets.

The technology will also help in terms of the industry's selection of patients for clinical trials and enable companies to identify any issues with compounds much earlier when it comes to efficacy and safety. So, the industry has much to gain by adopting AI and Machine Learning approaches. It can be used to good effect to build a strong, sustainable pipeline of new medicines[5].

REFERENCES

1. Metadata-based Auto-Programming Process https://www.lexjansen.com/phuse-us/2018/si/SI10.pdf
2. Power Data Explorer (PDE) - Data Exploration in an All-In-One Dynamic Report Using SAS & EXCEL https://www.lexjansen.com/phuse-us/2018/dv/DV13.pdf
3. Gartner, Augmented Analytics Is the Future of Data and Analytics, Rita L. Sallam, Cindi Howson, Carlie J. Idoine, 27 July 2017
4. https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html
5. https://www.drugtargetreview.com/article/15400/artificial-intelligence-drug-discovery
6. https://github.com/seatgeek/fuzzywuzzy
7. https://github.com/chartbeat-labs/textacy
8. https://spacy.io
9. https://www.nltk.org
10. http://www.antlr.org
11. https://en.wikipedia.org/wiki/Recurrent_neural_network
12. http://www.cs.cornell.edu/~kilian/papers/wmd_metric.pdf
13. https://spacy.io/models/en#en_vectors_web_lg
14. https://en.wikipedia.org/wiki/Levenshtein_distance