PhUSE 2014

Paper TS09

A world of clinical data at your fingertips

Mahdi About, Novartis, Basel, Switzerland

ABSTRACT
With the vast wealth of open online clinical data, we can look outwards to bring added value to our internal research. This paper aims to demonstrate how we can exploit these public domain data to answer scientific questions relating to our compounds. To do so, we will explore the technologies required to access this online data, eventually perform mashups with public API's, combine it with trial data, and present this information in the browser. In doing so, we hope to bring the world of online clinical data to the programmer's fingertips.

KEYWORDS
openFDA, ClinicalTrials.gov, PROC JSON, XML to SAS®, JSON to SAS®, SAS® to JSON, Multiple Correspondence Analysis MCA, PROC CORRESP, PROC CLUSTER, PROC GROOVY, HASH OBJECT.

DISCLAIMER
The opinions expressed in this paper (and in the related presentation and slides) are solely those of the presenter and not necessarily those of Novartis. Only public domain data has been used for analysis, Novartis does not guarantee the accuracy or reliability of the information provided herein.

INTRODUCTION
There is a world of data outside our windows, from junk to even the occasional gold nugget, but during the last decade, the majority of our industry has preferred to look at the curtains, blinded by the technical challenges and the regulatory limitations. However, things are changing: This started after the Food and Drug Administration Modernization Act (FDAMA) of 1997 and accelerated in 2013 after the "Data to knowledge to action" plan from the White House. The FDAMA produced various initiatives and led to the foundation of www.clinicaltrials.gov (CT.gov). NIH and the Food and Drug Administration worked together to develop the site, which was made available to the public in February 2000. The "Data to knowledge to action" plan, supported by semantic Web trends, is now producing outstanding initiatives like OpenFDA, a project initiated by the FDA's Office of Informatics, which aims to make the FDA's data accessible from a simple browser, avoiding the need for data requests through the Freedom of Information Act. It allows access to data such as adverse events, product labels and recalls. These initiatives represent challenges for clinical programmers as well as opportunities. In this paper we will expose some of the techniques associated with the exploitation of XML and JSON data, two of the formats most widely distributed over the internet.

CLINICALTRIALS.GOV
As an example, we decide to focus the first part of our case study on the exploitation of all oncology phase 2 to 3 studies within the last decade from industry filed in www.ClinicalTrials.gov.



Figure 1 - Screenshot ClinicalTrials.gov

After downloading and unzipping, we end up with 7697 studies (this number may change over time) covering oncology trials for adult and senior subjects, from 01/01/2004 to 01/01/2014. Each study is described by one unique XML file, encoded in UTF-8 characters.

XML files

We do not aim to create another tutorial to handle XML with XML Mapper/SAS®; the complete method has been covered in many valuable papers[1]. However, a brief refresher course in XML characteristics never hurts: XML stands for eXtensible Markup Language; XML files are human and machine readable files, self-described, hierarchical and internationalized. You can consult a study example following this link. CT.gov provides users with DTD and XSD (Schema) files, making our life easier with validated files and description of the data structure.

The easiest way to import the xml data (though not the most effective) would be to use the schema file to auto generate a map file using XML mapper. To proceed this way, you need XML mapper (freeware) and to follow a few steps:

1) Open XML mapper.
2) In File menu, select “Open XML Schema”.
3) Then click the “Automatically generate an XML map from the XML schema” icon (see figure 2).

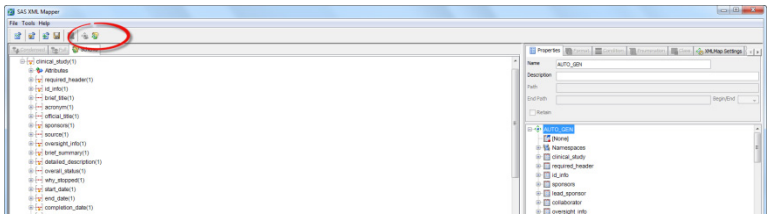


Figure 2 – XML Mapper generation of Map file

This will produce an XML map file which you can use along with the SAS® XML libname engine. However, this methodology leads to a very high number of datasets (i.e. 99) with coarse structure and data truncation. Note that for this case study, we need to process thousands of XML files, and to do so, one should keep in mind good programming practice and optimization. For instance, our first run to import data needed around ten hours to complete (while it takes a few minutes with the final macro and a proper mapping).

That said, the best solution remains manual mapping[1], specifying the dataset structure, length and format for each variable. We selected 32 variables mapped into one dataset, see Figure3 for map file preview.

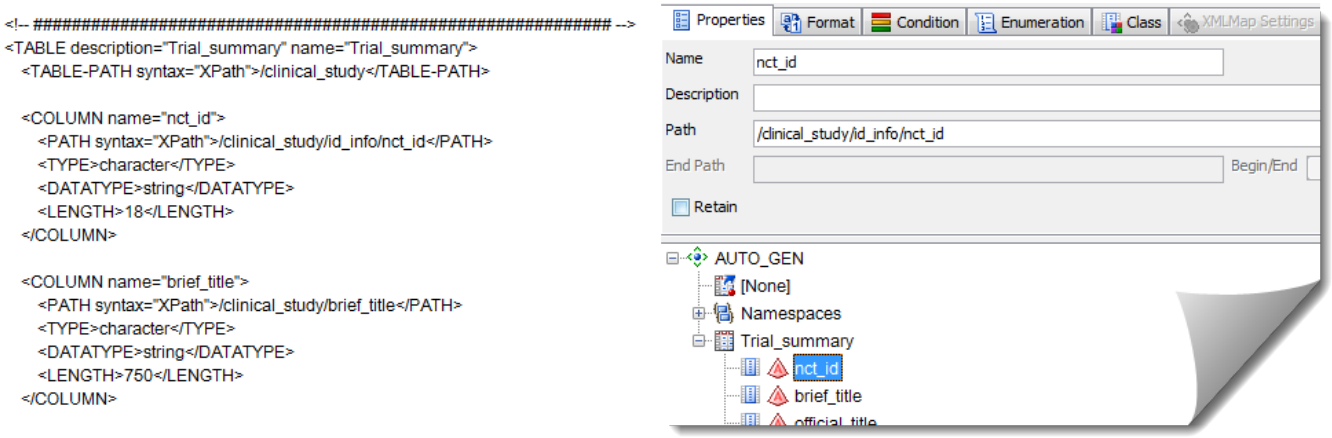


Figure 3 – Map file structure

Next step is to launch SAS® using the UTF-8 configuration file, you can edit your current sasv9.cfg to load the UTF-8 character set at program start, or incorporate the appropriate shortcut into start menu>All programs>SAS>Additional languages>SAS 9.x (Unicode support), or specify the appropriate config file with the –config parameter[2]:

```
-config "C:\Program Files\SAS\SASFoundation\9.3\nls\u8\sasv9.cfg"
```

Once SAS® is open, we declare all the XML files stored in our folder, by using a pipe filename (SAS® & OS communication).

```
*** Declare xml files;
filename indata pipe 'dir "C:\...\PHUse\cancer  Adult, Senior  Phase 2, 3  Industry  received
from 01012004 to 01012014 (7697 records)\*.xml" ' ;
```

The corresponding syntax with UNIX OS could be:

```
filename indata pipe "ls -t -R %sysfunc(pathname(XML_REP))/*.xml";
```

Next we will input the file list into a dataset:

```
*** List and count XML files present into the folder, store total number of files into fnum ;
data file_list ;
  retain cnt 0 ;
  length filen short_fname $50 ;
  infile indata truncover length=reclen;
  input ;
  if index(_infile_,'.xml')>0 then
       do ;
              filen = scan(_infile_,-1,' ') ;
              short_fname = tranwrd(filen,'.xml','') ;
              cnt + 1 ;
              output ;
       end;
  call symputx('fnum',cnt) ;
run;
```

At this step, we are in possession of all the XML file names and the count of XML files in the folder:

	cnt	filen	short_fname
1	1	NCT00075192.xml	NCT00075192
2	2	NCT00075218.xml	NCT00075218
3	3	NCT00075270.xml	NCT00075270
4	4	NCT00076011.xml	NCT00076011
5	5	NCT00076024.xml	NCT00076024
6	6	NCT00076310.xml	NCT00076310
7	7	NCT00076349.xml	NCT00076349

We can now access and input sequentially all the XML files into a single SAS® dataset by using a parameterized XML libname:

```
%do j=1 %to &fnum. ;
    *** Point on the n_th file and store its name into macrovariables ;
       data _null_ ;
              pointer = &j. ;
              set file_list point=pointer ;
              call symputx('fname',filen) ;
              call symputx('sfname',short_fname) ;
              putlog "NOTE: "filen= short_fname= ;
              stop ;
       run ;

       *** Declare XML file and MAP file associated;
       filename  SXLELIB "C:\...\PHUse\cancer  Adult, Senior  Phase 2, 3  Industry  received
from 01012004 to 01012014 (7697 records)\&fname." encoding="utf-8" ;
       filename  SXLEMAP "C:\...\PHUse\auto_gen2.map" ;
       libname   SXLELIB xmlv2 xmlmap=SXLEMAP access=READONLY;
```

From performance point of view, each iteration imports all available data, from which we see the importance of selecting an appropriate set of variables during the mapping, as one would do with the keep= option with a regular data step I/O.

When the first iteration executes, dataset(s) is (are) renamed, the next iterations will see the concatenation of the current iteration dataset(s) to the master dataset(s) with the append procedure:

```
*** Append current datasets to Master dataset ;
%do k=1 %to &numds. ;
      PROC APPEND base=_%scan(&dslist.,&k.) data=%scan(&dslist.,&k.) ;
      run ;
%end;
```

The advantage of this programming style is that your code is efficient whatever the number of datasets (1 or in our case, 99), as the dynamic syntax using the &dslist macro variable will take care of appending correctly all the similar datasets for you (if your XML files have the same structure).
Finally we end up with one dataset with the variables we selected at mapping time:

	nct_id	brief_title	official_title	agency	agency_class	source	authority
1	NCT00075192	CP-675,206 With Neoadjuvant Hormone Therapy in Patients With High Risk Prostate Cancer	A Phase 1, Open Label, Non-Randomized, Dose Escalation Study to Evaluate the Safety of CP-675,206 in Combination With Neoadjuvant Androgen Ablation and a Phase 2, Open Label, Randomized Study to Evaluate the Efficacy of CP-675,206 in Combination With Neoadjuvant Androgen Ablation and Androgen Ablation Alone in Patients With High Risk Prostate Cancer	AstraZeneca	Industry	AstraZeneca	United States: Food and Drug Administration
2	NCT00075218	A Study To Assess The Safety And Efficacy Of SU11248 In Patients With Gastrointestinal Stromal Tumor(GIST)	A Phase III, Randomized, Double-Blind, Placebo-Controlled Study Of SU011248 In The Treatment Of Patients With Imatinib Mesylate (Gleevec Tm, Glivec)-Resistant Or Intolerant Malignant Gastrointestinal Stromal Tumor	Pfizer	Industry	Pfizer	United States: Food and Drug Administration
3	NCT00075270	Paclitaxel With / Without GW572016 (Lapatinib) As First Line Therapy For Women With Advanced Or Metastatic Breast Cancer	A Randomized, Multicenter, Double-Blind, Placebo-Controlled, 2-Arm, Phase III Study of Oral GW572016 in Combination With Paclitaxel in Subjects Previously Untreated or Advanced or Metastatic Breast Cancer	GlaxoSmithKline	Industry	GlaxoSmithKline	United States: Food and Drug Administration
4	NCT00076011	Anti-angiogenesis Agent AG-013736 in Patients With Metastatic Renal Cell Carcinoma	Phase 2 Study of AG 013736 as Second-Line Treatment in Patients With Metastatic Renal Cell Cancer	Pfizer	Industry	Pfizer	United States: Food and Drug Administration

Figure 4 - Final clinical trials dataset

We are now in a position to be able to exploit ten years of oncology clinical trial data, which I have to say, is a rather exciting prospect. To be specific, the dataset contains 7967 records and 32 variables, characterizing 7967 Phase 1/2 to Phase III clinical trials conducted on 1461 421 adult patients suffering from 425 different neoplastic conditions.
These trials are distributed as follow over time:

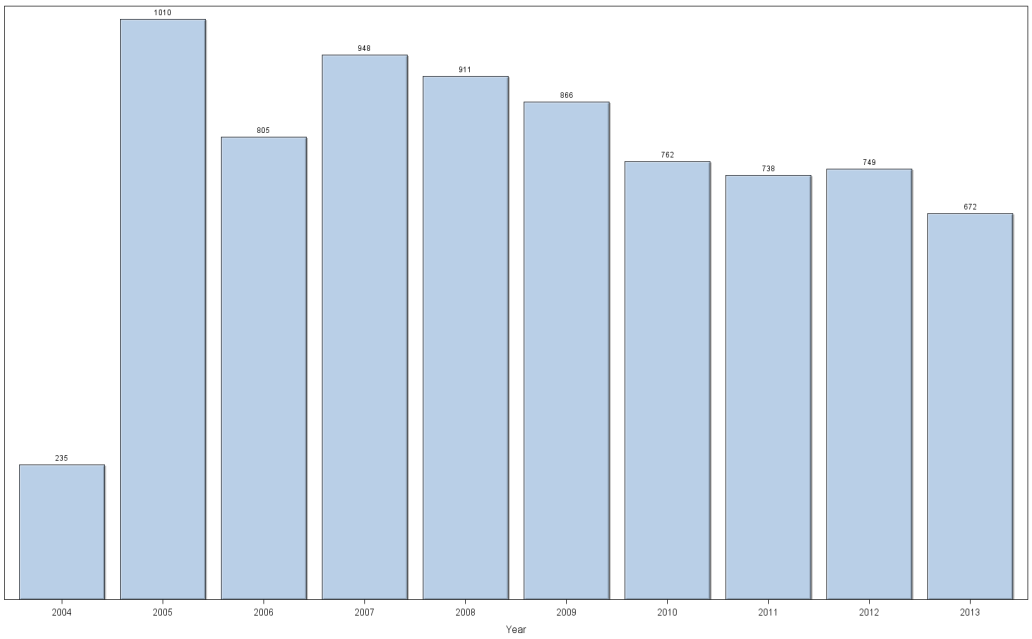


Figure 5 - Number of PI/II to PIII oncology trials from 2004 to 2013

The dramatic increase in the number of available trial data between 2004 and 2005 is explained by the fact that, in 2005, the newly introduced International Committee of Medical Journal Editors (ICMJE's) required that studies published in their journals had to be registered in clinicaltrials.gov. The 2007 Food and Drug Administration Amendments Act (FDAAA) imposed the registration of trials (drugs, biologics, and devices), which also contributed to a moderate increase in records[3]. Most of the studies investigate "drug" type treatment (6227 studies), follow by "biological" (600 studies), "devices" (65 studies) and finally "procedures" (58 studies). The rest is a mosaic of these different interventions combined.

Regarding the study design, it is interesting to see that during the last decade not all the sponsors used equally innovative study designs (i.e. phase combination: 1/2 and 2/3):

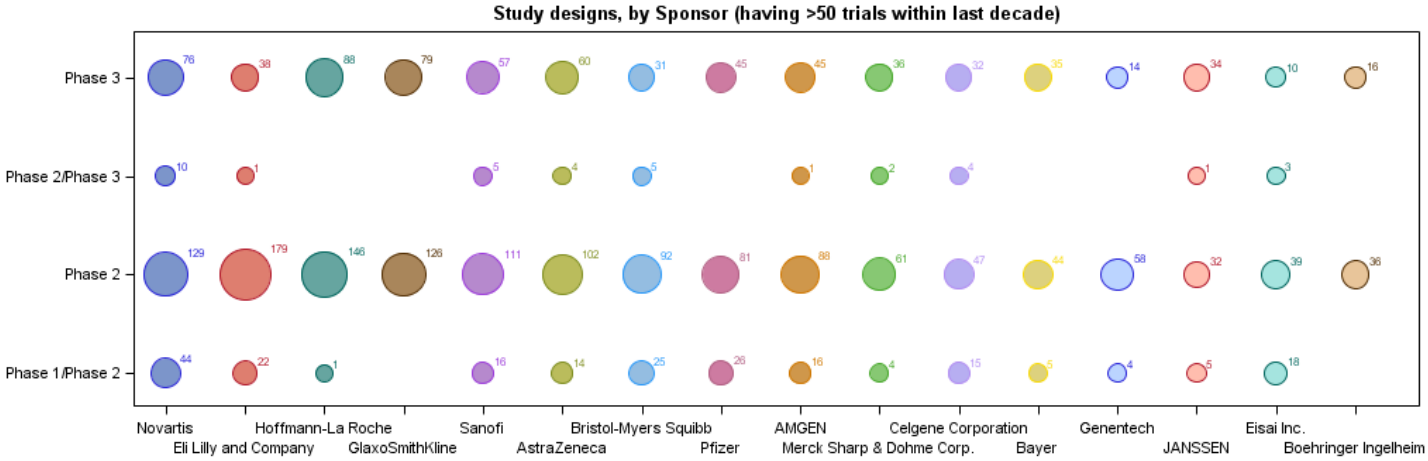


Figure 6 - Bubble plot of study designs, by sponsors

Sanofi, Eisai Inc, Celgene and Eli Lilly made important contributions in popularising these designs, but Novartis seems to be the biggest user of new clinical trial designs. This is encouraging for our profession: While we are never sure of the direction that clinical trials will take mid/long term, it is likely that design complexity will continue to increase, similar to the last decade, stimulated by progress in technology (wearable Health Devices) and biology ($100 patient full genomic sequence). This means more and more challenges for us!

EXPLORATORY ANALYSIS

"Multiple Correspondence Analysis (MCA) is a data analysis technique for nominal categorical data, used to detect and represent underlying structures in a data set. It does this by representing data as points in a low-dimensional Euclidean space. The procedure thus appears to be the counterpart of principal component analysis for categorical data. MCA can be viewed as an extension of simple correspondence analysis in that it is applicable to a large set of categorical variables" [4]. The method allows us to summarize a large amount of data in more comprehensible representations (like graphs) even if the interpretation of axis can sometimes be abstract and counter intuitive, especially when many columns contribute to one dimension (axis).

For our case study, we aim to characterize the dissimilarity between companies, i.e. the conditions they treat, and the way they treat them (drug, biological product, and device). Because of the vertical structure of the dataset, we generate a contingency table from PROC CORRESP. This can be done with the TABLES statement, or a combination of VAR/ID which requires more pre-processing. Nevertheless, we present below (and fully in the appendix) the two equivalent techniques to produce the MCA which is shown in figure 7.

We wanted to distinguish between different interventions used to treat the same condition . A big part of a company's 'DNA' is made up of the way in which they treat disease: immune-oncology, chemotherapy etc., so it is important to model these expertises. For that purpose, the CROSS=COLUMN is added as option. We chose to include only records from moderate to big companies for ease of readability.

Below we present two syntaxes producing the same result; this can help you if you have a different data structure:

```
/* Method 1 */
proc corresp data=trial_sum ALL outc=coord
observed outf=freqs dim=2 CROSS=COLUMN ;
where nbtrial ge 50 and
upcase(agency_class)="INDUSTRY" ;
tables source, condition intervention_type ;
run ;
```

```
/* Method 2 */
proc corresp data=DesignMatr2 dim=2
OUTC=coord2(rename=(source=_name_))  ;
where nbtrial ge 50 and
upcase(agency_class)="INDUSTRY"  ;
var _numeric_ ;
id source ;
run ;
```

The second method uses the design matrix: binary dummy variables are created for every class of a modal variable; please refer to code in the appendix for further details.

The MCA allows us to summarize a contingency table of all the clinical trials conducted on 196 neoplastic conditions, through 9 intervention types, by 16 companies during the last ten years in one comprehensible picture:

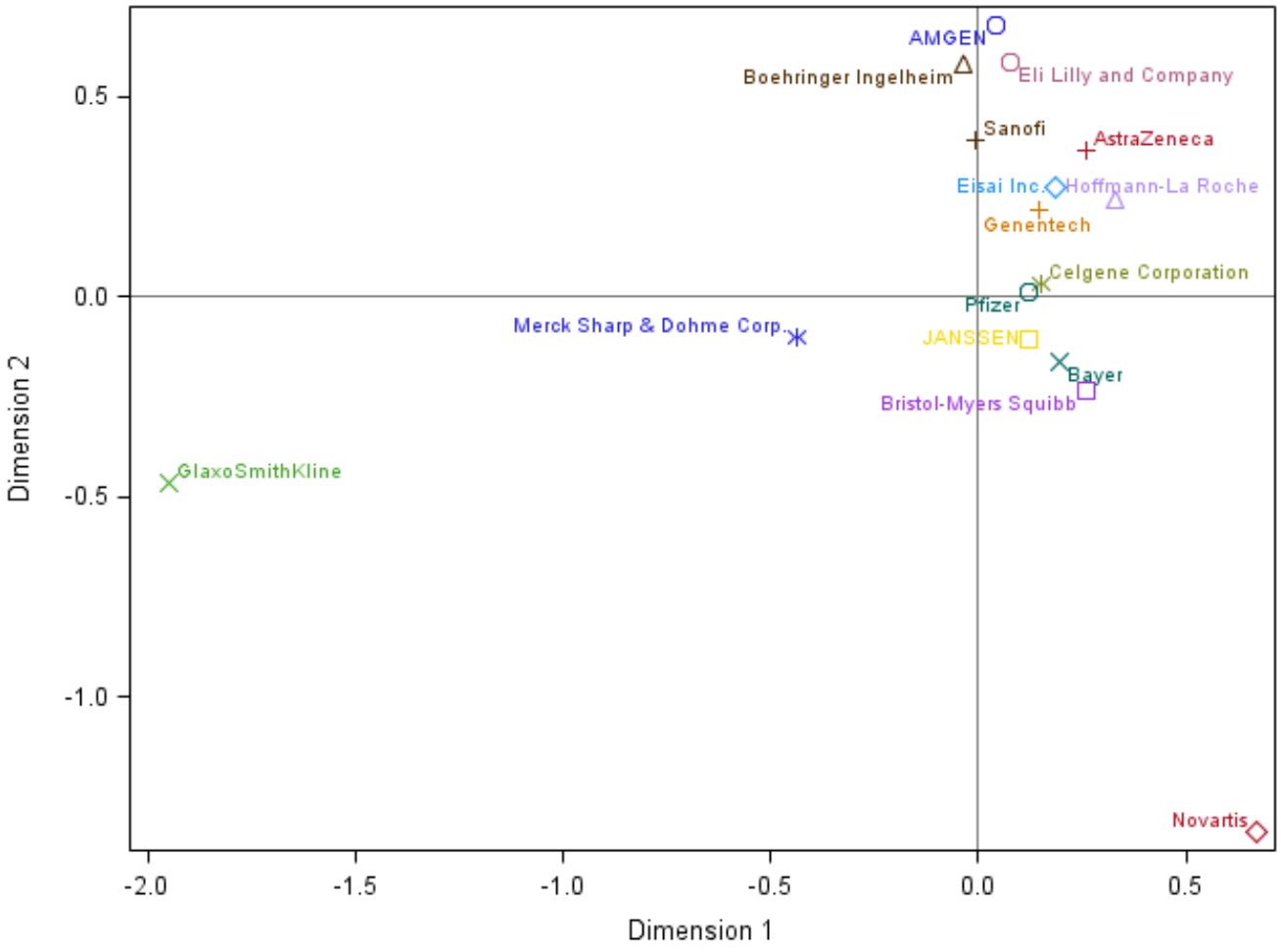


Figure 7 - MCA big pharma R&D

Each dimension compiles a part of the information relative to the variables included in the analysis. If you follow the news, it is very interesting to see the relative position of GSK and Novartis.
While it is easy to group observations when those are far from each other, it becomes more challenging when the points are very close or overlapping. To obtain a clear grouping, we will use PROC CLUSTER to create a valid clustering of companies with respect to their R&D dissimilarities.

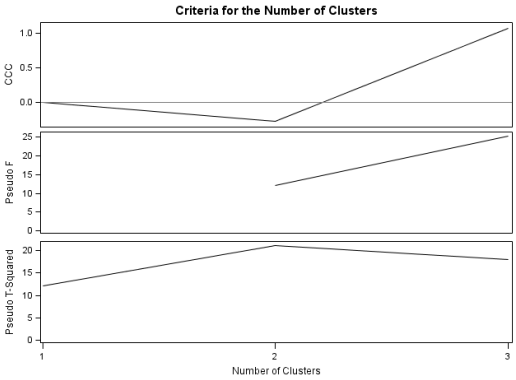


Figure 8 - Clustering criterion

The pseudo F and CCC suggest in figure 8 an optimal cluster number of 3. As stated in the SAS® documentation, relatively high values for Pseudo-F indicate a good number of clusters.

```
PROC CLUSTER DATA=coord METHOD=complete OUTTREE=D4 SIMPLE STD
RSQ CCC PSEUDO plots=ALL;
where _TYPE_ in ( 'OBS' );
VAR dim1 dim2 ;
ID  _name_ ;
run;

proc tree data=d4 nclusters=3 horizontal DISSIMILAR out=d4b;
id _name_ ;
run;
```

Finally the dendrogram (Figure 9) produced by PROC CLUSTER is given below. This suggests that during the last decade, more audacious strategies (i.e. pipeline development strategies) have been used by Novartis and GSK compared to the other companies. From the interpretation point of view, the longer the distance between two nodes the bigger the dissimilarity between two clusters.
To be explicit about the scope of this analysis, we focused only on diseases treated by a sponsor, and the means by which it was done, and we did not include whether the trial succeed or not, which, in the end, is the most important criteria for the patient. However, in our industry, investments made in R&D are correlated with the chance of success.

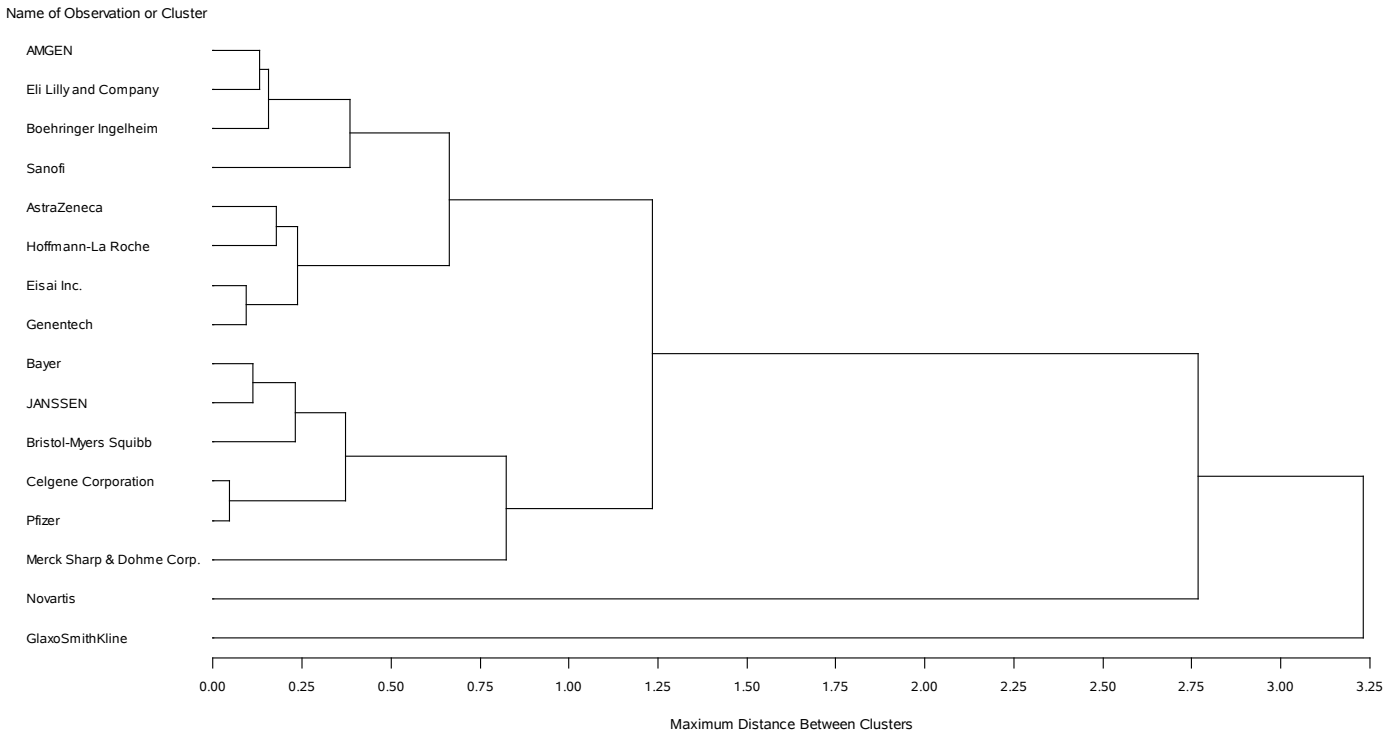


Figure 9 - Dendrogram

To conclude this part of the paper, we have used a real-life example to illustrate our technical demonstrations,linking theory to real possibilities. Beyond the SAS® code, there are meaningful analyses to be perform by a motivated and experienced programmer. Our next chapter will expose some of the technical challenges offered by the newer FDA initiatives and try to demonstrate how one can take advantage of this tremendous amount of data.

OPENFDA (BETA VERSION SEPT 2014)

"OpenFDA is an exciting new initiative from the Food and Drug Administration's Office of Informatics and Technology Innovation (OITI) spearheaded by FDA's Chief Health Informatics Officer (CHIO), Dr. Taha Kass-Hout, MD, MS. OpenFDA offers easy access to FDA public data and highlights projects using these data in both the public and private sector to further regulatory or scientific missions, educate the public, and save lives." [5]
Data is accessible through an API (Application Programming Interface), so you will need to familiarize yourself with the syntax before performing your first analysis. The API presents several possibilities: you can request listings or frequency tables, for many endpoints. The FDA is also thinking about making the data available for direct download. You can refer to this page to learn the API syntax.

All the queries start rather the same way, only the endpoint and the area differs:

Endpoint	Available for which areas	Corresponding syntax	Example of application
Adverse events	Drug Device	https://api.fda.gov/drug/event.json?search= https://api.fda.gov/device/event.json?search=	Reports over time, Who reports, Drug classes, Drug indications, Reported reactions
Labeling	Drug	https://api.fda.gov/drug/label.json?search=	Submission over time, routes of administration, drugs interactions
Enforcement reports	Drug Food Device	https://api.fda.gov/drug/enforcement.json?search= https://api.fda.gov/food/enforcement.json?search= https://api.fda.gov/device/enforcement.json?search=	Enforcement reports over time, who initiates, seriousness

After the equal sign you will enter your query using the predetermined fields at your disposal, which can be combined. For example, if you want to obtain the number of drug recalls over time you will enter this query in your browser address bar:

https://api.fda.gov/drug/enforcement.json?api_key=YOUR_API_KEY&count=report_date

To run this query, you need to replace YOUR_API_KEY by your registered API key (you can sign up here) or you can enter the below query, but the daily number of queries for unregistered user is much lower.

https://api.fda.gov/drug/enforcement.json?count=report_date

The query result appears now in your browser in the form of a JSON file:


```
{
  "meta": {
    "disclaimer": "openFDA is a beta research project and not for clinical use. Whi
    "license": "http://open.fda.gov/license",
    "last_updated": "2014-08-06"
  },
  "results": [
    {
      "time": "20040114",
      "count": 2
    },
    {
      "time": "20040121",
      "count": 14
    },
    {
      "time": "20040128",
      "count": 1
    },
    {
      "time": "20040204",
      "count": 2
```


You can save the file on your machine for archiving and/or processing and analysis.

JSON FILE PROCESS, SIMPLE CASE

JSON stands for JavaScript Object Notation. It is a lightweight and human readable data interchange format, just like XML. Despite the fact that JSON syntax is closely linked to java, it is a language independent format and can be used by many programmatic languages. Just like XML, JSON is "self-describing" and easy to understand, for machines and people.

Now let's imagine a famous fictional character, working for the *Princeton–Plainsboro hospital*, and unfortunately pain medication-dependent. Your colleague, the *Dr. James Wilson (House MD, Paul Attanasio)* worries a lot about your health, and is looking for strong arguments to make you decrease your consumption. For this, *Dr. Wilson* needs a reliable data source to

produce the listing of serious adverse events (MEDDRA Preferred Term) compiled for male patients from 1JAN2003 to 31DEC2013. So he politely asks to the over-worked *Princeton–Plainsboro hospital* statistical programmer to provide him with a frequency table, based on the OpenFDA data. The programmer writes the following query:

https://api.fda.gov/drug/event.json?api_key=YOUR_API_KEY&search=patient.drug.openfda.brand_name:%22vicodin%22+AND+serious:1+AND+patient.patientsex:1+AND+receivedate:[2003-01-01+TO+2013-12-31]&count=patient.reaction.reactionmeddrapt.exact

and saves the results in a file called *Gregory_house.json* (for some reason). Below are the first 3 records of the JSON file:

```
{
  "meta": {
    "disclaimer": "openFDA is a beta research project and not for clinical use. While we make
every effort to ensure that data is accurate, you should assume all results are
unvalidated.",
    "license": "http://open.fda.gov/license",
    "last_updated": "2014-08-06"
  },
  "results": [
    {
      "term": "PAIN",
      "count": 433
    },
    {
      "term": "DYSPNOEA",
      "count": 404
    },
    {
      "term": "NAUSEA",
      "count": 382
    },
```

Because the *Plainsboro hospital* statistical programmer is familiar with self-described data formats, he writes a short program to input the JSON data in a SAS® dataset:

```
filename house "C:\Users\...\Desktop\PHUse\gregory_house.json" ;

data greg(drop=tot) ;
    infile house lrecl = 2048 truncover scanover;
       input
      @'"term": ' PT_TXT $75.
      @'"count": ' freq best. ;
         retain tot 0 ;
              if cmiss(freq)=0 then tot+freq ;
              PT_TXT = translate(PT_TXT,'','" ,');
       call symputx('tot_freq',tot) ;
       call execute("data greg ; set greg ;percent = '('||put(freq /
&tot_freq.,percent8.1)||')' ;run;") ;
run ;
```

Thanks to the SCANOVER function, SAS® parses the infile until a character @'character-string' expression is found.
This produces the desired table:

Prefered Term	Frequency	Percentage
PAIN	433	(2.8%)
DYSPNOEA	404	(2.7%)
NAUSEA	382	(2.5%)
FATIGUE	353	(2.3%)
ANXIETY	337	(2.2%)
ASTHENIA	334	(2.2%)
DEPRESSION	331	(2.2%)
MYOCARDIAL INFARCTION	307	(2.0%)

This simple JSON file is very easy to parse and to import into SAS®. Unfortunately, JSON files can be much more complex. They can contain values, arrays or objects, sometimes (often) nested.

JSON FILE PROCESS, COMPLEX CASE

Unfortunately, there is no native solution provided by SAS® to parse JSON file with SAS 9.4. To illustrate a solution, we will process a rather complex hierarchical JSON file. This file lists the 5 first records of serious adverse events experienced by male patient between 2012-01-01 and 2013-12-31. Each variable can contain either a value, an array containing values, or objects:

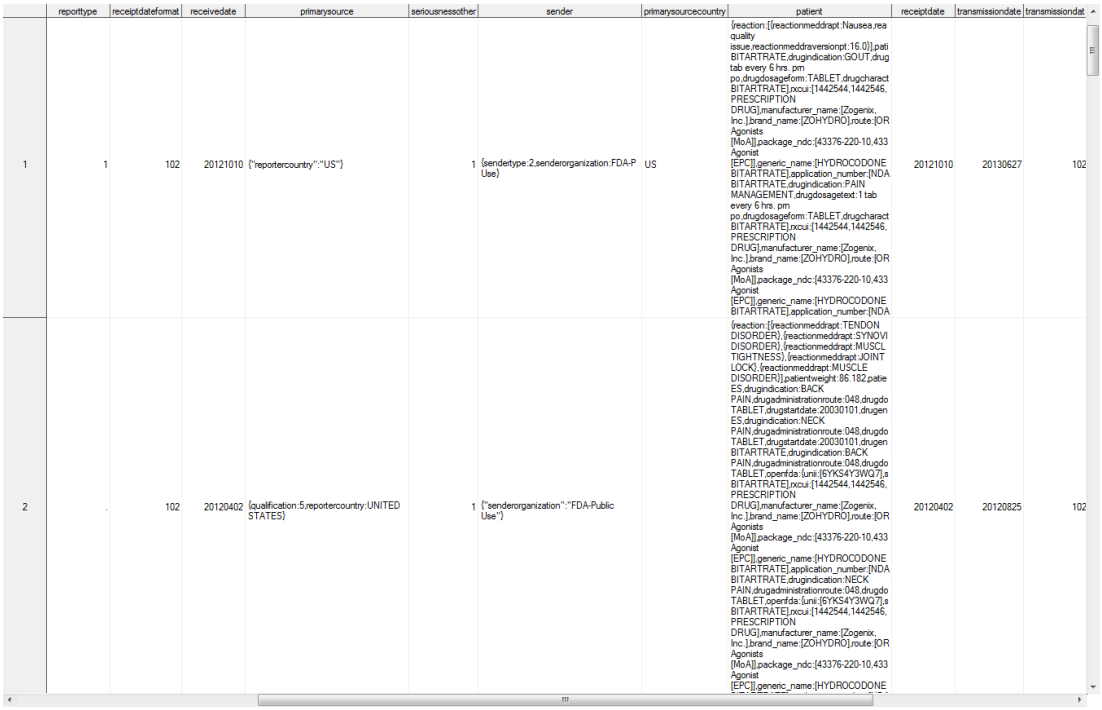
```
"results": [
    {
      "safetyreportid": "8834456",
      "fulfillexpeditecriteria": "2",
      "receiver": {
        "receivertype": "6",
        "receiverorganization": "FDA"
      },
      "receivedateformat": "102",
      "reporttype": "1",
      "receiptdateformat": "102",
      "receivedate": "20121010",
      "primarysource": {
        "reportercountry": "US"
      },
      "seriousnessother": "1",
      "sender": {
        "sendertype": "2",
        "senderorganization": "FDA-Public Use"
      },
      "primarysourcecountry": "US",
      "patient": {
        "patientonsetage": "61",
        "reaction": [
          {
            "reactionmeddrapt": "Nausea",
            "reactionmeddraversionpt": "16.0"
          },
          {
            "reactionmeddrapt": "Dizziness",
            "reactionmeddraversionpt": "16.0"
          },
          {
            "reactionmeddrapt": "Product quality issue",
            "reactionmeddraversionpt": "16.0"
          }
        ],
```

Without a JSON parsing function, process this kind of text file quickly becomes complicated, even using advanced PERL regular expressions. We first need to transform this vertical structure file to a SAS® horizontal table. To do so, we will create a small JAVA program to transform the JSON file to CSV and we will call the execution of the JAR file from SAS® by using call routines (the java program is available in the appendix):

```
data _null_;
call system ('cd C:\temp');
call system ('java -jar json2csv.jar
"https://api.fda.gov/drug/event.json?search=patient.drug.openfda.generic_name:%22hydrocodone%
22+AND+serious:1+AND+receivedate:[2012-01-01+TO+2013-12-31]&limit=10"
C:\temp\hydrocodone.csv');
call system ('pause');
run ;
```

With the source code of the program, you might be able to run the program within PROC GROOVY, but because of security limitation on my machine, I was not able to process it in this way. Once this step is done, you can use PROC IMPORT to import and process the CSV file:

```
proc import out=work.event datafile="C:\temp\hydrocodone.csv" dbms=csv replace;
  getnames=yes;
run;
```



	reporttype	receiptdateformat	receivedate	primarysource	seriousnessother	sender	primarysourcecountry	patient	receiptdate	transmissiondate	transmissiondat
1	1	102	20121010	{"reportercountry":"US"}	1	{sendertype:2,senderorganization:FDA-Public Use}	US	{reaction:[{reactionmeddrapt:Nausea,reaquality issue,reactionmeddraversionpt:16.0}],patiBITARTRATE,drugindication:GOUT,drug tab every 6 hrs ,prn po,drugdosageform:TABLET,drugcharactBITARTRATE],rxcui:[1442544,1442546,PRESCRIPTION DRUG],manufacturer_name:[Zogenix, Inc.],brand_name:[ZOHYDRO],route:[OR Agonists [MoA],package_ndc:[43376-220-10,433 Agonist [EPC]],generic_name:[HYDROCODONE BITARTRATE],application_number:[NDABITARTRATE,drugindication:PAIN MANAGEMENT,drugdosagetext:1 tab every 6 hrs ,prn po,drugdosageform:TABLET,drugcharactBITARTRATE],rxcui:[1442544,1442546,PRESCRIPTION DRUG],manufacturer_name:[Zogenix, Inc.],brand_name:[ZOHYDRO],route:[OR Agonists [MoA],package_ndc:[43376-220-10,433 Agonist [EPC]],generic_name:[HYDROCODONE BITARTRATE],application_number:[NDA	20121010	20130627	102
2		102	20120402	{qualification:5,reportercountry:UNITED STATES}	1	{"senderorganization":"FDA-Public Use"}		{reaction:[{reactionmeddrapt:TENDON DISORDER},{reactionmeddrapt:SYNOVI DISORDER},{reactionmeddrapt:MUSCL TIGHTNESS},{reactionmeddrapt:JOINT LOCK},{reactionmeddrapt:MUSCLE DISORDER}],patientweight:86.182,patieES,drugindication:BACK PAIN,drugadministrationroute:048,drugdoTABLET,drugstartdate:20030101,drugenES,drugindication:NECK PAIN,drugadministrationroute:048,drugdoTABLET,drugstartdate:20030101,drugenBITARTRATE,drugindication:BACK PAIN,drugadministrationroute:048,drugdoTABLET,openfda:{uni:[6YKS4Y3WQ7],sBITARTRATE],rxcui:[1442544,1442546,PRESCRIPTION DRUG],manufacturer_name:[Zogenix, Inc.],brand_name:[ZOHYDRO],route:[OR Agonists [MoA],package_ndc:[43376-220-10,433 Agonist [EPC]],generic_name:[HYDROCODONE BITARTRATE],application_number:[NDABITARTRATE,drugindication:NECK PAIN,drugadministrationroute:048,drugdoTABLET,openfda:{uni:[6YKS4Y3WQ7],sBITARTRATE],rxcui:[1442544,1442546,PRESCRIPTION DRUG],manufacturer_name:[Zogenix, Inc.],brand_name:[ZOHYDRO],route:[OR Agonists [MoA],package_ndc:[43376-220-10,433 Agonist [EPC]],generic_name:[HYDROCODONE	20120402	20120825	102

Figure 10 - JSON to CSV file imported in SAS

Once the data was accessible to SAS®, we figured out that name/value pairs are easily extracted, while arrays and JSON Object are extracted as text block from the source JSON file. The data has a regular structure, which allows the use of PERL regular expressions for extraction. Please note that it is mandatory to know about lazy repetition factors[6]. Indeed, because of the nested structure, using greedy syntax will match the entire text block, which in our case is useless for further processing. The important line is this one:

```
re1 = PRXPARSE("/\w+:\[*.+?(,|\})/io") ;
```

This line will extract a text block matching VARIABLE_NAME:SOME_TEXT(ended by a coma or a bracket):

```
found=reaction:[{reactionmeddrapt:UNDERDOSE}
found=reactionmeddrapt:ENERGY INCREASED}
found=reactionmeddrapt:BLOOD PRESSURE DECREASED}
found=reactionmeddrapt:INCORRECT DRUG ADMINISTRATION DURATION}
found=reactionmeddrapt:DRUG DEPENDENCE}
found=patientonsetage:67,
found=patientonsetageunit:801,
```

From time to time, the program will also catch the array of objects declaration line:
`found=reaction:[{reactionmeddrapt:UNDERDOSE},..`

Smart post processing can get rid of this problem, for instance when you extract variable name and value, you can use the scan function negative count feature (bold lines):

```
 _start = 1 ;
 _stop = length(patient) ;
  call prxnext(re1, _start, _stop, patient, position, length);
do while (0 < position <= length(patient) );
   found = substr(patient, position, length);
        putlog found= ;
        variablelab = upcase(scan(found,-2,':')) ;
        valuev = scan(found,-1,':') ;
        if cmiss(VALUEV,VARIABLELAB)=0 then output ;
   call prxnext(re1, _start, _stop, patient, position, length);
end;
```

Finally, you obtain your variable in vertical structure:

variablelab	valuev
REACTIONMEDDRAPT	Nausea
REACTIONMEDDRAVERSIONPT	16.0
REACTIONMEDDRAPT	Dizziness
REACTIONMEDDRAVERSIONPT	16.0
REACTIONMEDDRAPT	Product quality issue
REACTIONMEDDRAVERSIONPT	16.0
PATIENTONSETAGE	61
PATIENTWEIGHT	105.24
PATIENTONSETAGEUNIT	801
DRUGADDITIONAL	1

After transposition, you are in possession of vertical structure dataset, much easier to process in your beloved SAS to produce a masterpiece of an output which will make your friend quit hydrocodone:

PRODUCT_TYPE	REACTIONMEDDRAPT	ROUTE	RXCUI
HUMAN PRESCRIPTION DRUG	TOXICITY TO VARIOUS AGENTS	ORAL	1442544
HUMAN PRESCRIPTION DRUG	ENCEPHALITIS	ORAL	1442544
HUMAN PRESCRIPTION DRUG	CRIME	ORAL	1442544
	DRUG ABUSER		
	HOMICIDE		
	IMPRISONMENT		
	PRESCRIPTION FORM TAMPERING		
	THEFT		

OTHER OPTIONS FOR IMPORTING JSON: PROC GROOVY

This way of processing, by calling a java program from the console, might scare some people who are more comfortable with running the complete process in SAS itself. For this purpose, you still have the choice writing a program with PROC GROOVY. Falko Schulz provides an example here. I think this solution is a bit less flexible, because you need to adapt your code to the structure of your JSON file. Furthermore, depending your SAS version, you may need to update your groovy JAR files. However, using groovy generates simpler code, and requires less complex post processing:

```
/*  produce a CSV file */
proc groovy;
add classpath="C:\temp\groovy-all-2.2.1.jar";
add classpath="C:\temp\opencsv-2.3.jar";

submit "C:\Users\...\PHUse\gregory_house.json" "C:\Users\...\PHUse\greg.csv";
import groovy.json.*;
import au.com.bytecode.opencsv.CSVWriter

def input = new File(args[0]).text;
def output = new JsonSlurper().parseText(input);
```

```
def csvoutput = new FileWriter(args[1]) ;
CSVWriter out = new CSVWriter(csvoutput);

String[] header = new String[2];
    header[0] = "Preferred term";
    header[1] = "Freq";
    out.writeNext(header);

output.results.each {
       println it.term ;
       String[] content = new String[2];
       content[0] = it.term.toString();
       content[1] = it.count.toString();
       out.writeNext(content)
}
out.flush();
endsubmit;
quit;
```

This program produces the corresponding CSV table, with less code, and less post processing:

Preferred term	Freq
PAIN	433
DYSPNOEA	404
NAUSEA	382
FATIGUE	353
ANXIETY	337
ASTHENIA	334

This is a good alternative for simple JSON files, but requires some (basic) knowledge in groovy language, furthermore the code needs to be adapted for every new files.

A MORE CONCRETE EXAMPLE FOR CLINICAL DEVELOPMENT

What if we were able to decrease the work load of clinical teams? If by chance, the FDA were to provide drug interaction data, we might be able to programmatically spot possible drug interactions in concomitant medication or inclusion criteria datasets. Well it is surely our lucky day, because open FDA provides drug-drug interaction (DDI) data with OpenFDA label API (and raw datasets are probably coming up).

Now imagine you are part of a team conducting a clinical trial on Methotrexate pharmacokinetics. For this project you are short of resources and you would like to automate the detection of DDI at recruitment or during the trial. First you need to obtain the list of drugs which may interact with Methotrexate:

https://api.fda.gov/drug/label.json?search=drug_interactions:methotrexate&count=openfda.generic_name.exact

Secondly, as previously, we need to transform the JSON file into CSV:

```
data _null_;
  call system ('cd C:\temp');
  call system ('java -jar json2csv.jar
"https://api.fda.gov/drug/label.json?search=drug_interactions:methotrexate&count=openfda.gene
ric_name.exact" "C:\temp\methotrexate.csv"');
  call system ('exit');
run ;


proc import out=work.ddi_metho(rename=(term=PT_TXT)) datafile="C:\temp\methotrexate.csv"
dbms=csv replace;
   getnames=yes;
run;
```

We also create a dummy dataset for testing purposes:

```
data conco;
   input patid visit PT_TXT $12. ;
   datalines;
 10 1 IBUPROFEN
 10 2 ALOE VERA
 11 1 PARACETAMOL
 11 1.1 FOLIC ACID
 12 1 IMATINIB
 14 1 RETINOL
;
run ;
```

And finally we iterate through the concomitant dataset with the Hash iterator searching for records having a preferred term matching with the drug-drug interaction dataset:

```
data _null_ ;
   if 0 then set conco ;
   if _N_ = 1 then do;

      /* Read conco data set and store in asc order in hash obj */
      declare hash CONCO(dataset:"work.conco", ordered: 'yes');
      CONCO.defineKey('PT_TXT');
      CONCO.defineData('PT_TXT','pat','vis');
      declare hiter iter('CONCO');
      CONCO.defineDone();

      /* Define variables PT_TXT as key and data for hash object */
      declare hash DDI(dataset:"work.ddi_metho", ordered: 'no');
      DDI.defineKey('PT_TXT');
      DDI.defineData('PT_TXT');
      DDI.defineDone();

      /* Avoid uninitialized variable notes */
      call missing(pat, vis, PT_TXT);
   end;

/* Retrieve PT_TXT values in ascending order */
rc = iter.first();
   do while (rc = 0);
              /* Find hash object keys with corresponding preferred term */
              if DDI.find(key:PT_TXT) = 0 then putlog PT_TXT= PAT= VIS= ;
              rc = iter.next() ;
   end;
run;
```

Which produces the below messages in the log:

```
NOTE: There were 101 observations read from the data set WORK.DDI_METHO.
PT_TXT=FOLIC ACID pat=11 vis=1.1
PT_TXT=IBUPROFEN pat=10 vis=1
```

We chose to perform a lookup with the hash iterator. Often, concomitant medication datasets are quite big. In-memory processing allows rapid iteration through big datasets without impacting too much the shared resource (server). Nevertheless for inexact matches, hash objects may not be the most accurate technique, as an index method is missing in SAS® hash for partial match detection, like in this case, for retinol.

EXPORTING YOUR RESULTS TO JSON FORMAT

Data visualization is a very trendy topic on the web today. You can now find tools developed solely for data representation and graphics (like D3.js). Many of these tools are developed in JavaScript, so they natively support JSON format. After complex analysis or data processing with SAS®, we might need to export the results as a JSON file for data visualization or reporting. To illustrate this new possibility newly offered by PROC JSON[7] in SAS 9.4® we will use the data extracted from ClinicalTrials.gov's XMLs and convert those to JSON format:

```
%macro JSONGEN()  ;

proc sql noprint ;
create table dims as select distinct _name_, dim1, dim2 from  work.coord where _type_ = 'OBS'
order by _name_ ;
select distinct '"'||strip(_name_)||'"', strip(_name_), count(distinct _name_) into:listq
separated by ';',:listnq separated by '#',:cnt TRIMMED from work.dims ;
quit;

proc json out='C:\temp\trials.json' pretty  TRIMBLANKS  NOSASTAGS ;
%do i=1 %to &cnt. ;
      write values %scan(&listq.,&i.,;) ;
      write open array ;
            export trials(where=(source in (%scan(&listq.,&i.,;))) keep=stycode source
phase condition beginy2 healthy_volunteers intervention_type) ;
      write close ;
%end ;
      write values "Coordinates" ;
      write open array ;
            export dims ;
      write close ;
run ;

%mend JSONGEN ;
%JSONGEN ;
```

1. OUT determines where to export the JSON file.
2. PRETTY creates a more readable format.
3. TRIMBLANKS trim the character variables.
4. NOSASTAGS removes the metadata.
5. WRITE VALUES hardcode the value determined by the loop.
6. OPEN ARRAY open an array of values, an array starts with "[".
7. EXPORT writes variables from a dataset.
8. WRITE CLOSE closes the opened array.
9. WRITE VALUES writes the value "Coordinates".
10. WRITES the values contained into dims dataset.

Finally we obtain the JSON file below. This file is natively usable by many languages, and can be easily exported to the cloud for data visualization:


```
{
  "AMGEN": [
    {
      "stycode": "NCT00083616",
      "source": "AMGEN",
      "phase": "Phase 2",
      "condition": "Neoplasms, Second Primary",
      "beginy2": 2004,
      "healthy_volunteers": "No",
      "intervention_type": "Biological"
    },
    {
      "stycode": "NCT00089635",
      "source": "AMGEN",
      "phase": "Phase 2",
      "condition": "Neoplasm Metastasis",
      "beginy2": 2004,
```

```
    "healthy_volunteers": "No",
    "intervention_type": "Drug"
  },
  {
```


CONCLUSION

Web technologies are advancing rapidly, more rapidly than our industry, and one thing is for sure; the amount of data will continue to grow exponentially. We have tried to present in this paper a useful example, stretching from data acquisition to final result, based on a real online repository. We believe these methods are relevant not only to pharma, but also to other sectors such as insurance, finance, mergers-and-acquisitions, research etc. The US government and FDA plan to make more and more databases available to everyone, opening a very exciting time for clinical R&D. Furthermore, private companies are also acting: Eli Lilly, Pfizer and Novartis launched an initiative to improve the quality of the data provided through CTs.gov. They aim to ease the access of clinical trials to patients. All these synergetic efforts, in conjunction with initiatives like the blue button will allow patient to join a study with a simple click.
Because of all these coming challenges, we will need more innovative programmers and simpler tools to access this wealth of information. We hope future version of SAS will see JSON parsing tools as well as PROC JSON improvements: One can even imagine that, in near future, statistical programmers will not produce reports anymore, but will export JSON data to the cloud, where standard reporting webpages will display the data interactively.

REFERENCES

1. http://www2.sas.com/proceedings/sugi29/119-29.pdf,
   https://support.sas.com/resources/papers/proceedings12/220-2012.pdf,
   http://www2.sas.com/proceedings/forum2008/099-2008.pdf.

2. http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#a000104286.htm

3. http://www.ncbi.nlm.nih.gov/books/NBK50886/

4. http://en.wikipedia.org/wiki/Multiple_correspondence_analysis

5. https://open.fda.gov/about/

6. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a003288497.htm

7. http://support.sas.com/documentation/cdl/en/proc/67327/HTML/default/viewer.htm#p0ie4bw6967jg6n1iu629d40f0by.htm

ACKNOWLEDGMENTS

With thanks to Hong Yan, Andy Chopra, Danny Tuckwell, Karma Tarap and Jean-Francois Gimenez.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mahdi ABOUT
Novartis Oncology
WSJ-103.3.10.17, Novartis Pharma AG
Postfach CH-4002 Basel, SWITZERLAND
mahdi.about@novartis.com
www.novartis.com



APPENDIX

IMPORT OF XML FILES

```
%macro input_xml() ;

*** Set options ;
option mprint mlogic bufsize=0 ;

*** CLEAN WORK lib ;
proc datasets lib=WORK kill memtype=data FORCE NOWARN NODETAILS NOLIST;
run;quit ;

*** Declare xml files, and folder to store imported data ;
filename indata pipe 'dir "C:\Users\...\PHUse\cancer  Adult, Senior  Phase 2, 3  Industry
received from 01012004 to 01012014 (7697 records)\*.xml" ' ;

*** List and count XML files present into the folder, store number of files into fnum ;
data file_list ;
  retain cnt 0 ;
  length filen short_fname $50 ;
  infile indata truncover length=reclen;
  input ;
  if index(_infile_,'.xml')>0 then
       do ;
              filen = scan(_infile_,-1,' ') ;
              short_fname = tranwrd(filen,'.xml','') ;
              cnt + 1 ;
              output ;
       end;
  call symputx('fnum',cnt) ;
run;

*** sort the files by names ;
proc sort data = file_list ;
       by filen ;
run ;

*** CLEAR the pipe filename ;
filename indata clear ;

*** LOOP on the number of XML files into folder to transfer each into datasets ;
%do j=1 %to &fnum. ;

    *** Point on the n_th file and store its name into macrovariables ;
       data _null_ ;
              pointer = &j. ;
              set file_list point=pointer ;
              call symputx('fname',filen) ;
              call symputx('sfname',short_fname) ;
              stop ;
       run ;

       *** Declare XML file and MAP file associated ;
       filename  SXLELIB "C:\Users\...\PHUse\cancer  Adult, Senior  Phase 2, 3  Industry
received from 01012004 to 01012014 (7697 records)\&fname." encoding="utf-8" ;
       filename  SXLEMAP "C:\Users\...\PHUse\auto_gen2.map" ;
       libname   SXLELIB xmlv2 xmlmap=SXLEMAP access=READONLY;

       *** Preprocess during the first iteration ;
       %if %sysevalf(&j.=1) %then %do ;
```

```
            *** Copy content from XML file to work lib and export name of datasets into
member_work ds ;
            proc datasets lib=SXLELIB NODETAILS NOLIST ;
                  contents directory data=_all_ out=work.member_work(keep=memname) noprint
;
                  copy in=SXLELIB out=work ;
            run;
            quit ;

            *** Get the name list of datasets and their total number ;
            proc sql noprint ;
                  select distinct strip(memname), count(distinct memname) into:dslist
separated by ' ',:numds TRIMMED from member_work ;
            quit ;

            *** Rename datasets to keep from one iteration to an other ;
            proc datasets lib=WORK NODETAILS NOLIST ;
            change
                  %do k=1 %to &numds. ;
                        %scan(&dslist.,&k.) = _%scan(&dslist.,&k.)
                  %end;
            ;
            run;
            quit ;

      %end ;
      %else %do ;
            *** Copy content from XML file to work lib ;
            proc datasets lib=SXLELIB NODETAILS NOLIST ;
                  copy in=SXLELIB out=work ;
            run;
            quit ;

            *** Apprend current datasets to Master dataset ;
            %do k=1 %to &numds. ;
                  proc append base=_%scan(&dslist.,&k.) data=%scan(&dslist.,&k.) ;
                  run ;
            %end;
            *** Delete datasets except master datasets ;
            proc datasets lib=WORK memtype=data NODETAILS NOLIST ;
            save
                  %do k=1 %to &numds. ;
                        _%scan(&dslist.,&k.)
                  %end;
                  file_list member_work
            ;
            run;
            quit ;

      %end ;
      %symdel fname sfname / NOWARN ;
%end;

** Store Final datasets into permanent lib ;
%do k=1 %to &numds. ;
      data perm.%scan(&dslist.,&k.) ; set work._%scan(&dslist.,&k.) ; run ;
%end;

*** Clear filenames ;
filename _all_ clear ; libname _all_ clear ;
%mend input_xml ;
%input_xml ;
```

MULTIPLE CORRESPONDANCE ANALYSIS AND CLUSTERING

```
/* MCA: chose method 1  */
ods graphics on / ANTIALIAS=ON  HEIGHT=1050px
width=1680px border=off;
title 'Correspondence analysis of oncology
R%nrstr(&)D';
title2 'for big Pharmas (>than 50 trials
during last decade) ';
ods output RowContr=RowContrib
ColContr=ColContr ;
proc corresp data=trial_sum ALL outc=coord
observed outf=freqs dim=2 CROSS=COLUMN ;
where Sponsor_Num_trials ge 50 and
upcase(agency_class)="INDUSTRY"  ;
tables source,condition intervention ;
run ;


/* end method 1*/
```

```
/* Or MCA method 2  */
* Production of design matrix (binary vars);
PROC TRANSREG DATA = trial_sum DESIGN NOPRINT
NOINT CPREFIX=5 ;
MODEL CLASS (condition*intervention / ZERO =
NONE)  ;
id source agency_class;
OUTPUT OUT = DesignMatr(drop=_TYPE_ _NAME_
intercept condition intervention) ;
RUN ;
options NOQUOTELENMAX ;
%let nbvar = %sysfunc(countw("&_TRGIND.")) ;
* Extraction of variable names  ;
proc sql noprint;
      select count(distinct name) into:nbvar
TRIMMED from dictionary.columns where
memtype="DATA" and memname="DESIGNMATR" and
type="num"
;quit;

proc sort data=DesignMatr ;
      by source ;
run ;
* frequency table generation ;
data DesignMatr2 ;
      set DesignMatr ;
      by source ;

      array nums [*] _numeric_ ;
      array ret [&nbvar.] ret1-ret&nbvar. ;

      do i=1 to dim(nums) ;
            retain ret 0 ;
            if first.source then ret[i]=0 ;
            if nums[i]=1 then ret[i]+1 ;
            if last.source then
nums[i]=ret[i] ;
      end;
      nbtrial = sum(of condi:) ;
      if last.source ;
      drop ret: i intervention ;
run ;
* MCA  ;
proc corresp data=DesignMatr2 dim=2
OUTC=coord(rename=(source=_name_))  ;
where nbtrial ge 50 and
upcase(agency_class)="INDUSTRY"  ;
var _numeric_ ;
id source ;
run ;
/* end method 2 */
```

```
* Displaying coordinates on scatterplot  ;
proc sgplot data=coord noautolegend ;
where _TYPE_ in ( 'OBS' );
xaxis label="Dimension 1" ;
yaxis label="Dimension 2" ;
refline 0 / axis=x ;
refline 0 / axis=y ;
scatter x=dim1 y=dim2 / datalabel=_NAME_ group=_name_ MARKERATTRS=(size=9) ;
run ;
title;
```

```
* clustering points using complete algorithm ;
PROC CLUSTER DATA=coord METHOD=complete OUTTREE=D4 SIMPLE STD RSQ CCC PSEUDO plots=ALL;
where _TYPE_ in ( 'OBS' );
VAR dim1 dim2 ;
ID  _name_ ;
run;
* Generate the dendrogram  ;
proc tree data=d4 nclusters=3 horizontal DISSIMILAR out=d4b;
id _name_ ;
run;
```

CONVERT JSON TO CSV, JAVA PROGRAM SOURCE CODE

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.nio.charset.Charset;
import org.json.CDL;
import org.json.JSONArray;

public class json2csv {

       public static void main(String[] args) {
                     try{

                        String fichierJson =args[0];
                        String fichierCsv =args[1];
                        String jsonString="";
                        if (args[0].contains("http"))
                        {
                                InputStream is = new URL(fichierJson).openStream();
                                BufferedReader rd = new BufferedReader(new
InputStreamReader(is, Charset.forName("UTF-8")));
                                StringBuilder sb = new StringBuilder();
                              int cp;
                              while ((cp = rd.read()) != -1) {
                                sb.append((char) cp);
                              }
                             jsonString=sb.toString();
                             System.out.println("JSON URL readed ok");
                        }
                        else
                        {
                                  InputStream ips=new FileInputStream(fichierJson);
                                  InputStreamReader ipsr=new InputStreamReader(ips);
                                  BufferedReader br=new BufferedReader(ipsr);
                                  String ligne;
                                  while ((ligne=br.readLine())!=null){
                                         jsonString+=ligne;
                                  }
                                  br.close();


                             System.out.println("JSON readed ok");
                         }
                                  jsonString=jsonString.substring(jsonString.indexOf("["));
                                   JSONArray docs = new JSONArray(jsonString);
```

```
                                    String csv = CDL.toString(docs);

                                    File file=new File(fichierCsv);

                                    FileWriter fw = new FileWriter (file);
                                    BufferedWriter bw = new BufferedWriter (fw);
                                    PrintWriter fichierSortie = new PrintWriter (bw);
                                    fichierSortie.println (csv);
                                    fichierSortie.close();
                                    System.out.println(" file "+ fichierCsv +" has been
created!");
                             }
                      catch (Exception e){
                             System.out.println(e.toString());
                      }

       }


}
```

CONVERT JSON TO CSV, PROC GROOVY EXAMPLE

```
/*  produce a CSV file */
proc groovy;
add classpath="C:\temp\groovy-all-2.2.1.jar";
add classpath="C:\temp\opencsv-2.3.jar";

submit "C:\Users\...\PHUse\gregory_house.json" "C:\Users\...\PHUse\greg.csv";
import groovy.json.*;
import au.com.bytecode.opencsv.CSVWriter

def input = new File(args[0]).text;
def output = new JsonSlurper().parseText(input);
def csvoutput = new FileWriter(args[1]) ;
CSVWriter out = new CSVWriter(csvoutput);

String[] header = new String[2];
    header[0] = "Preferred term";
    header[1] = "Freq";
    out.writeNext(header);

output.results.each {
       println it.term ;
       String[] content = new String[2];
       content[0] = it.term.toString();
       content[1] = it.count.toString();
       out.writeNext(content)
}
out.flush();
endsubmit;
quit;
```

CONVERT JSON TO CSV WITH JAVA, THEN IMPORT AND POST PROCESSING IN SAS

```
* Executing java program from SAS  ;
data _null_;
call system ('cd C:\temp');
call system ('java -jar json2csv.jar "C:\temp\hydrocodone.json" "C:\temp\hydrocodone.csv"');
call system ('pause');
run ;
* CSV import  ;
proc import out=work.event datafile="C:\temp\hydrocodone.csv" dbms=csv replace;
  getnames=yes;
run;
```

```
* Parse text looking for nested JSON values  ;
data event2 ;
       set event end=eof  ;

       if _N_ = 1 then do ;
              * Using lazy repetition factors  ;
              re1 = PRXPARSE("/\w+:\[*.+?(,|\})/io") ;
       end ;
       retain re1 ;
       length found variablelab valuev $100 ;

       receiverorg = scan(receiver,2,':,');
       receivertype = scan(receiver,4,':,');
       duplicatesource = scan(reportduplicate,2,':,');
       duplicatenumb = scan(reportduplicate,4,':,');
       qualification = scan(primarysource,2,':,');
       reportercountry = scan(primarysource,4,':,');
       sendertype = scan(sender,2,':,');
       senderorganization = scan(sender,4,':,');
       * parsing whole text field looking for pattern  ;
       _start = 1 ;
       _stop = length(patient) ;
        call prxnext(re1, _start, _stop, patient, position, length);
      do while (0 < position <= length(patient) );
         found = substr(patient, position, length);
              putlog found= ;
              variablelab = upcase(scan(found,-2,':')) ;
              valuev = scan(found,-1,':') ;
                    array chars {*} receivertype duplicatenumb reportercountry
senderorganization variablelab valuev qualification ;
                           do i=1 to dim(chars) ;
                                  chars[i] = strip(translate(chars[i],'','{},[]')) ;
                           end;
                    drop receiver reportduplicate primarysource sender _: patient found
position length i re1;
              if cmiss(VALUEV,VARIABLELAB)=0 then output ;
         call prxnext(re1, _start, _stop, patient, position, length);
      end;
run ;
* dynamically generate by statement for future transposition  ;
proc sql noprint;
       select distinct upcase(name) into:namelst separated by " " from dictionary.columns
where memtype="DATA" and memname="EVENT2"
             and upcase(name) not in ("VALUEV" "VARIABLELAB") ;
;
quit;

proc sort data=event2 out=event3 ; by safetyreportid VARIABLELAB VALUEV   ; run;
* Set a proper record ID for transposition (several AEs per report) ;
data event4 ;
       set event3 ;
       by  safetyreportid VARIABLELAB VALUEV   ;
       retain recid 1 ;
       if first.safetyreportid or first.VARIABLELAB then recid=1 ;
       if not first.VARIABLELAB then recid+1 ;
run ;

proc sort data=event4 ;by &namelst. recid ;run ;
* Transposition vertical to honrizontal  ;
proc transpose data=event4 out=event_t(drop=_name_) let ;
       var VALUEV ;
       id VARIABLELAB ;
       by  &namelst. recid ;
```

```
run ;
```

EXPORT SAS DATASET AS A JSON FILE

```
%macro JSONGEN()  ;
* Get the list of company and generate corresponding macrovariables ;
proc sql noprint ;
create table dims as select distinct _name_, dim1, dim2 from  work.coord where _type_ = 'OBS'
order by _name_ ;
select distinct '"'||strip(_name_)||'"', strip(_name_), count(distinct _name_) into:listq
separated by ';',:listnq separated by '#',:cnt TRIMMED from work.dims ;
quit;
* Dynamic generation of JSON file  ;
proc json out='C:\...\trials.json' pretty TRIMBLANKS nosastags ;
%do i=1 %to &cnt. ;
       write values %scan(&listq.,&i.,;) ;
       write open array ;
              export trials(where=(source in (%scan(&listq.,&i.,;))) keep=stycode source
phase condition beginy2 healthy_volunteers intervention_type) ;
       write close ;
%end ;
       write values "Coordinates" ;
       write open array ;
              export dims ;
       write close ;
run ;

%mend JSONGEN ;
%JSONGEN ;
```