PharmaSUG China 2019 - Paper DS-037

A Simple Aggregated Web Page to Collect Trial Summary Domain Dictionaries

Jingwei Huang, Pfizer (China) Research & Development Co. Ltd, Shanghai, China

ABSTRACT

Trial Summary (TS) domain is considered as an essential domain to be included in submission. Populating specific TS parameters requires correct controlled terminologies. Programmers have to navigate through various webpages to obtain external information for TS parameters. Once the webpage interface changes, extra time would be spent in figuring out the correct pathway. To centralize information on clinical trials, a customized aggregated web page can be established based on Python. This paper would introduce how Python packages, i.e. BeautifulSoup, could be applied for web scraping and how Python package, i.e. Django, for interactive web page development.

INTRODUCTION

All Trial Design Model (TDM) should be included in SDTM submission (1). TS domain, which is one key domain in TDM, allows sponsor to summary a trial in a structured format (2). It has been clearly indicated in SDTMiG v3.2 or SDTMiG v3.3 Appendix Section whether one TS parameter should be included in domain. As suggested by Kristin Kelly et al (3), populating specific TS domain parameters is like putting a puzzle. Even though programmers may not have Interdisciplinary knowledge to fulfill the entire TS domain parameters correctly by themselves, a draft version should be provided to work with both clinicians and statisticians efficiently. That means parameters including REGID (Registry Identifier), TRT (Investigational Therapy or Treatment), CURTRT (Current Therapy or Treatment), COMPTRT (Comparative Treatment Name), PCLAS (Pharmacological Class of Invest. Therapy) and INDIC (Trial Disease/Condition Indication) should be prompted with correct TSVAL (Parameter Value) or TSVALCD (Parameter Value Code). Therefore, at least five external webpages or dictionaries should be navigated to fetch information for TS domain. There may be long time intervals between one submission and another submission. It is quite a cognitive burden for programmers to learn how to use these webpages given that interface for specific webpage would change and memory for TS would fade.

To simplify the process for searching dictionary values, a python-based, customized webpage could be established to centralize above information in one page. Python is an object-oriented programming language. Various types of packages could be found online to construct programmers' own project given python's fastest-growing programmer community. To establish such a web page, technical route based on Django-Requests-BeautifulSoup would be used. Django package is a Python-based free and open-source web framework, which allows programmers to develop webpage user interface from a semi-finished web product. Requests and BeautifulSoup packages are excellent third-party packages to scratch and parse web information into structured data for further programming.

This paper would introduce how to use Python packages (Requests and BeautifulSoup) to build a python crawler to obtain web page information in backend and the basic steps to set up web interface (frontend) by Django package.

SUMMARY OF TS RELATED DICTIONARIES AND EXTERNAL LINKS

As indicated Kristin Kelly et al (3), six TS parameters would involve with five external web or dictionaries (Table 1). TSVAL for REGID should be found in ClinicalTrials.gov (https://clinicaltrials.gov/) if one study has registered in FDA for submission. Besides, if one study was initiated before 2008, there might be not such an ID for it. TSVALCD for TRT, CURTRT and COMPTRT should be searched in Substance Registration System, Unique Ingredient Identifier (UNII) (https://fdasis.nlm.nih.gov/srs/srs.jsp). TSVAL for PCLAS should be fetched from FDA Established Pharmacologic Class Text Phrases for Indications and Usage (https://www.fda.gov/drugs/laws-acts-and-rules/plr-requirements-prescribing-information). TSVALCD for PCLAS should be obtained from National Drug File Reference Terminology (NDF-RT) (https://ncit.nci.nih.gov/ncitbrowser/pages/multiple_search.jsf?nav_type=terminologies). TSVALCD for

INDIC should be received from Systematized Nomenclature of Medicine -- Clinical Terms (SNOMED). Although NDF-RT and SNOMED share the same link, their exact entries should be opened by clicking the corresponding text under given link. Their exact entry links would be updated along with their dictionary version.

TSPARMCD	TSPARM	TSVCDREF	Inclusion in TS	External Link
REGID	Registry Identifier	ClinicalTrials .gov	Required	https://clinicaltrials.gov/
COMPTRT	Comparative Treatment Name	UNII	If Applicable	https://fdasis.nlm.nih.gov/srs/srs.jsp
CURTRT	Current Therapy or Treatment	UNII	Conditionally Required	https://fdasis.nlm.nih.gov/srs/srs.jsp
TRT	Investigational Therapy or Treatment	UNII	Conditionally Required	https://fdasis.nlm.nih.gov/srs/srs.jsp
PCLAS	Pharmacological Class of Invest. Therapy	NDF-RT	Conditionally Required	https://www.fda.gov/drugs/laws-acts-and-rules/plr-requirements-prescribing-information and https://ncit.nci.nih.gov/ncitbrowser/pages/multiple_search.jsf?nav_type=terminologies
INDIC	Trial Indication	SNOMED	If Applicable	https://ncit.nci.nih.gov/ncitbrowser/pages/multiple_search.jsf?nav_type=terminologies

Table 1. Summary of TS Related Dictionaries and External Links

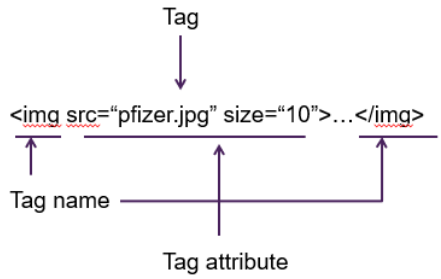
SOFTWARE

Entire project was performed using Python 3.5 (Python Software Foundation) (4). Django (Version 2.2.1) (Django Software Foundation) was used to develop web framework in Python (5). Requests (Version 2.10.0) and BeautifulSoup (Version 4.4.1) were deployed to create a Python crawler for fetching and parsing information online (6,7). Installation of Python and Python packages would not be described in this paper.

AGGREGATED WEBPAGE

WEBPAGE SOURCE

In general webpages are organized by information tag or JavaScript (JS) or combination of both. In terms of information tag, there are three general forms for it. They are eXtensible Markup Language (XML), JavaScript Object Notation (JSON) and YAML Ain't Markup Language (YAML). Below is an example for xml file (Display 1.). Text editor could be used for viewing xml source code. As seen, one completed xml code is surrounded by paired information tag <img> and </img>. If there are certain attributions for tag or text inside tag, they can be added after first tag. For those xml codes without text description inside, single tag ends with "/" could be created.



If no element in "…"

Display 1. An Example for XML.

Html file is one special format from xml file. Below is general structure for html file (Display 2.). Required tag <html> would wrap up all html codes and contents at the very beginning and the very end. Tag <head> and tag <body> would be used if there are corresponding elements within header and body of html.

```
<html>
   <head>
        xxxx
   </head>
   <body>
        <p class="title"> … </p>
   </body>
</html>
```

Display 2. General Structure for HTML.

PYTHON CRAWLER

A Python crawler is a Python script that crawls web information according to certain rules. There are two common packages for web crawling. One is Requests package, the other is Scrapy package. Requests package is a lightweight crawler suitable for webpage-level scratching meanwhile Scrapy package is a website-level professional framework. Since current requirement for TS domain is based on certain webpage only, Requests is preferred and much easier to learn and use. By far, webpage embedded with JS has been out of discussion scope. Currently, webpages for REGID, UNII and PCLAS are comprised by information tag while webpages for NDF-RT and SNOMED are involved with JS. Therefore, only the most recent links for NDF-RT and SNOMED would be provided in the final product.

Requests and BeautifulSoup Package

As described before, Requests package can be used for developing lightweight crawler. There is a generic code framework for Requests as following:

```
import requests

def getHTMLText(url):
    try:
          r=requests.get(url, timeout=30)
          r.raise_for_status()
          r.encoding=r.apparent_encoding
          return r.text
    except:
          return "Connection Error"
```

Here, one function called getHTMLText is created which is similar to sas macro function. The only parameter for this function is URL. Once getHTMLText function is used, text content of response object would be returned to assigned object (Display 3.). That means the source html file coding would be returned for further programming.

```
In [6]: outtext=getHTMLText("https://www.baidu.com/")
   ...: print(outtext)
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible
content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css
href=https://ss1.bdstatic.com/5eN1bjq8AAUYm2zgoY3K/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head> <body
link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img
hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=//www.baidu.com/s
class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8>
<input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span
class="bg s_ipt_wr"><input id=kw name=wd class=s_ipt value maxlength=255 autocomplete=off autofocus=autofocus></span><span class="bg
s_btn_wr"><input type=submit id=su value=百度一下 class="bg s_btn" autofocus></span> </form> </div> </div> <div id=u1> <a
href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=https://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a
href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>视频</a> <a
href=http://tieba.baidu.com name=tj_trtieba class=mnav>贴吧</a> <noscript> <a
href=http://www.baidu.com/bdorz/login.gif?login&amp;tpl=mn&amp;u=http%3A%2F%2Fwww.baidu.com%2f%3fbdorz_come%3d1 name=tj_login
class=lb>登录</a> </noscript> <script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+
encodeURIComponent(window.location.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+ '" name="tj_login" class="lb">
登录</a>');
                </script> <a href=//www.baidu.com/more/ name=tj_briicon class=bri style="display: block;">更多产品</a> </div> </div>
</div> <div id=ftCon> <div id=ftConw> <p id=lh> <a href=http://home.baidu.com>关于百度</a> <a href=http://ir.baidu.com>About Baidu</a>
</p> <p id=cp>&copy;2017 Baidu <a href=http://www.baidu.com/duty/>使用百度前必读</a>  <a href=http://jianyi.baidu.com/
class=cp-feedback>意见反馈</a> 京ICP证030173号  <img src=//www.baidu.com/img/gs.gif> </p> </div> </div> </div> </body>
</html>
```

Display 3. Usage and return from request generic code framework.

Afterwards, BeautifulSoup can be introduced to parse returned text content. It does the tree traversal stuff for programmers and converts html file source codes into structured data with following:

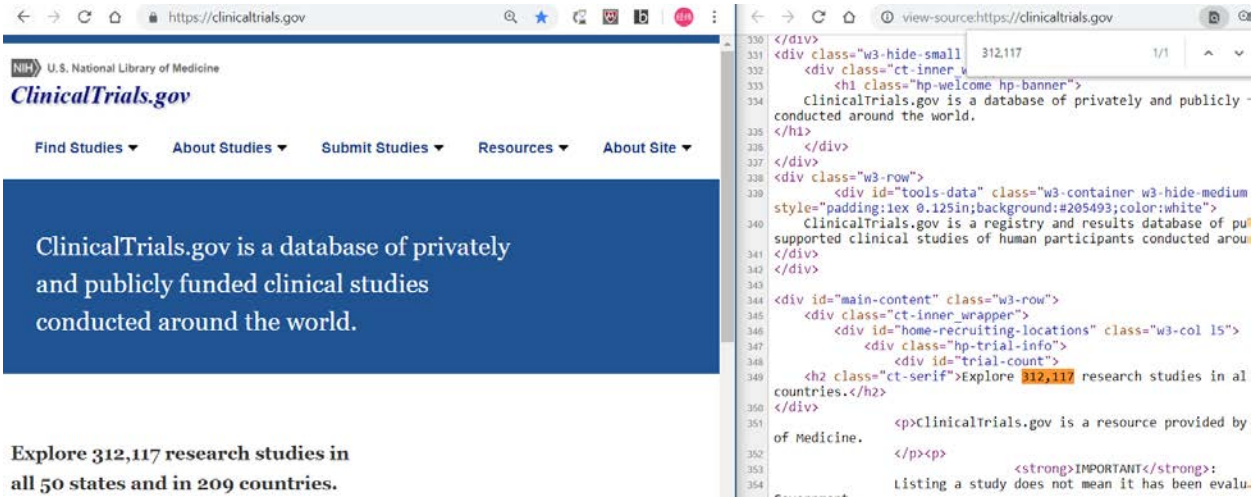
```
from bs4 import BeautifulSoup

soup=BeautifulSoup(outtext,"html.parser")
```

Outtext here is the returned result from Display 3.. One BeautifulSoup object would be returned as a result. There are several object methods that can be used for further programming. **Find_all** method and **find** method are the most useful ones (Table 2) Find_all method can look through a tag's descendants and retrieve all descendants that match filters even though certain tag is nested within other tags. Find method would stop after finding the first result which would be more efficient.

Methods	Details
Soup.find_all(tag_name,xxx)	The find_all() method looks through a tag's descendants and retrieves *all* descendants that match your filters.
Soup.find(tag_name,xxxx)	The find_all() method scans the entire document looking for results, but sometimes you only want to find one result. If you know a document only has one <body> tag, it's a waste of time to scan the entire document looking for more.

Table 2. Summary of two BeautifulSoup Object Methods



Display 4. Example for retrieving value from clinicaltrials.gov.

Above is a snapshot from https://clinicaltrials.gov/ (Display 4.). The left part of it displays the user interface for this page. The right part displays the source behind this page which could be opened by right clicking on the page and selecting "View page source" in Chrome browser. If we want to retrieve how many research studies are in the world, we would find what is the tag for this number. According to the right part of snapshot, it is found that tag <h2> with class attribution of value "ct-serif" is used to wrap up target number. Therefore, the first step to find out target number is typing the following code:

```
Soup.find_all('h2',attrs={"class","ct-serif"})
```

Then one listing would be returned for further programming (Output 1.).

```
[<h2 class="ct-serif">Explore 307,343 research studies in all 50 states and
in 210 countries.</h2>,
 <h2 class="ct-serif hp-constituent">Patients and Families</h2>,
 <h2 class="ct-serif hp-constituent">Researchers</h2>,
 <h2 class="ct-serif hp-constituent">Study Record Managers</h2>]
```

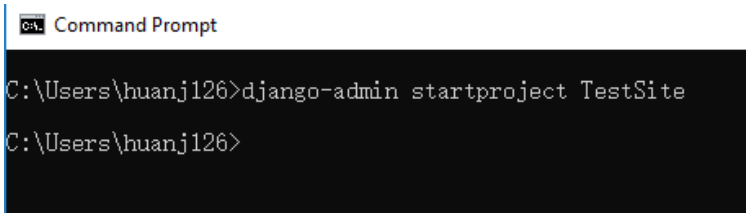
Output 1. Output from manipulating BeautifulSoup object with find_all method

Django Package

Django is a Python-based free and open-source web framework, which follows the model-template-view (MTV) architectural. Some well-known sites including Instagram, Youtube, Dropbox and Mozilla use Django. For programmers who know Python well, a Django-based webpage is not necessary indeed. However, a Django-based webpage can involve those who don't have basic knowledge for Python into cooperation easily. This paper would not go into details for how MTV works in Python rather than giving general steps for setting up a simple Django web.

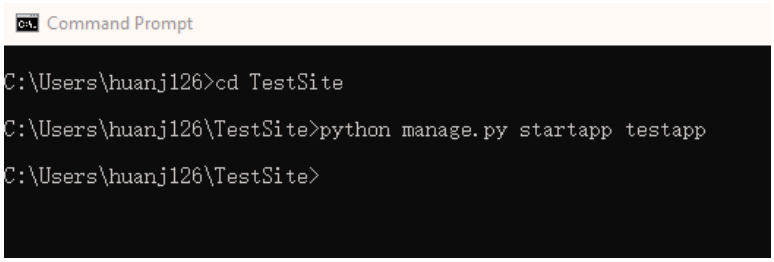
Below is 9 general steps for Django:

1. Open windows command line by pressing windows key, typing "cmd.exe" and pressing Enter. Then type "Django-admin startproject yoursite" to create a project (Display 5.).



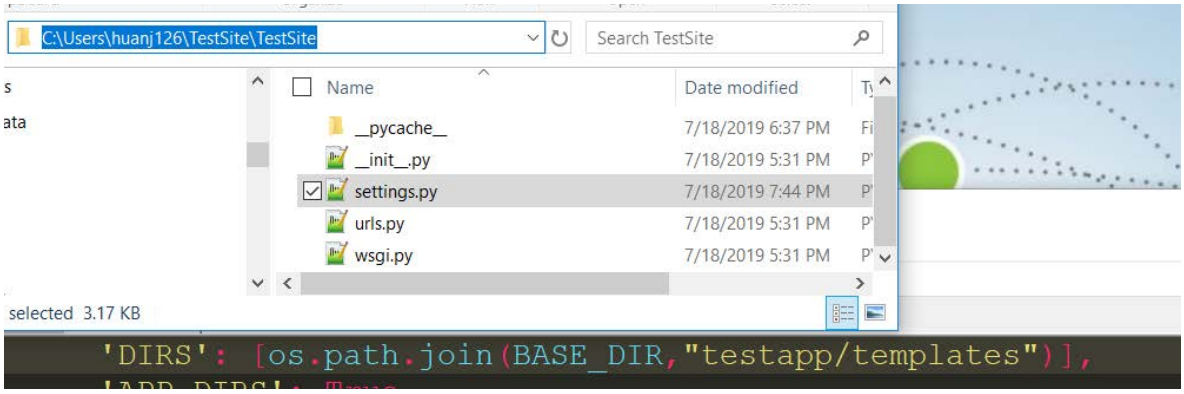
Display 5. First Step for Django.

2. Switch into your site by typing “cd your_site”. And type “python manage.py startapp your_app” to create your app (Display 6.).



Display 6. Second Step for Django.

3. Open location path which is ./your_site/your_site folder and open setting.py to modify line 57 as below in which testapp is your app name and templates is the folder you want to place your designed html user interface (Display 7.).

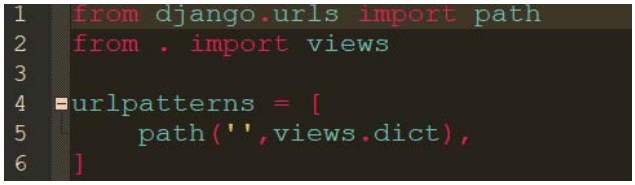


Display 7. Third Step for Django.

4. Open urls.py in the same folder and modify file as below to include urls file under your app (Display 8.). Line 17 and line 20 are modified from template.

Display 8. Fourth Step for Django.

5. Open location path which is ./your_site/your_app, open urls.py, import views and add path for function which would be defined in views file (Display 9.).



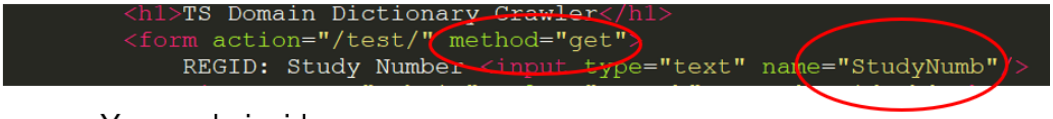
Display 9. Fifth Step for Django.

6. Create a folder named templates which is the same name as step 3 and put your designed UI (Appendix A) into it.

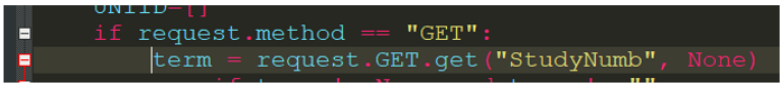
7. To get input from webpage, html file tag should have defined method attribution and relevant input

name attribution (Display 10.). Then if statement and request.Get.get method should be used to obtain value from web input.

- Your input would have a name attribution.



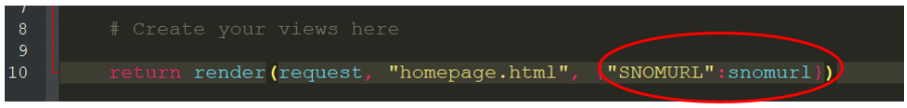
- Your code inside:



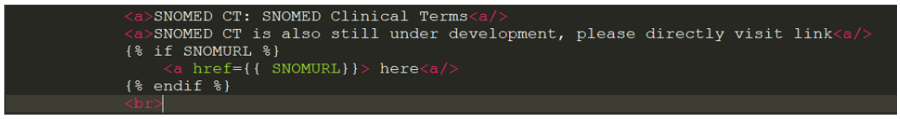
Display 10. Seventh Step for Django.

8. Back to your app location, open views.py file and add your code inside (Appendix B).
9. To pass output back to html file, Django Template Language (DTL) would be used which is wrapped by "{% " and " %}" (Display 11.). For example, updated link for SNOMED would be passed by a string data in Appendix B. If statement would be used to judge whether this data passed with name of "SNOMURL" is empty. If not, corresponding html code would be prompted in user interface.

- Code in Appendix B.

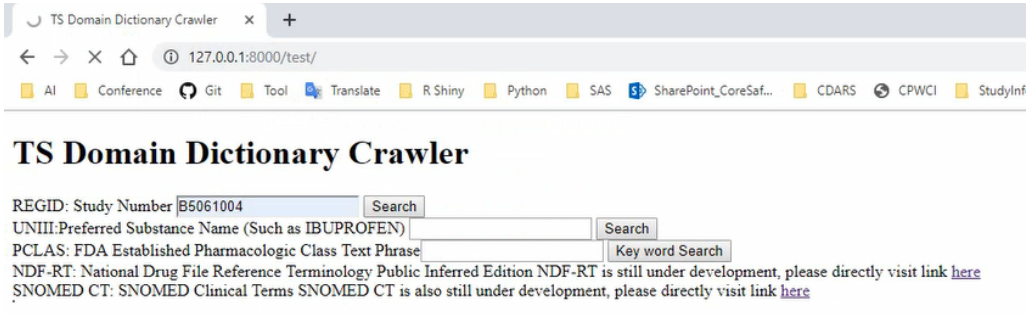


- Html file in Appendix A.



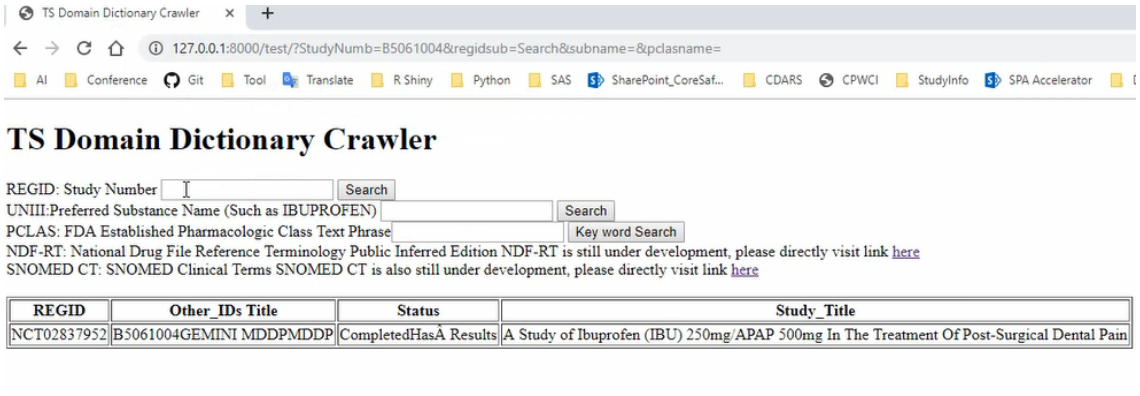
Display 11. Ninth Step for Django.

FINAL PRODUCT



Display 12. User Interface for Aggregated TS webpage.

When Django is settled well, programmers can access this webpage through browser by typing '127.0.0.1:8000/test" given specific link defined in step 4 of Django setup (Display 12.). Each line represents either a search for specific TS dictionary value or external link for searching TS dictionary value. For example, to search REGID for study B5061004, this study number can be inserted into the first text window after REGID of first line. Then button "Search" can be clicked to start searching. It would take several seconds to populate its RREGID (Display 13.). To help programmer decide whether search result is correct, supplementary columns are provided with additional information.



Display 13. User Search Output from Aggregated TS webpage.

CONCLUSION

Automation programming is an inevitable trend for entire drug programming industry. Given python's flexibility and high operability, Python might become one of the most powerful tools for innovation. At this paper, a python-based aggregated webpage is introduced to collect necessary TS domain dictionary value in one page. It is estimated by author that it can save a programmer for at least one hour in one submission TS creation. In the future, it is desired that one aggregated webpage could be provided to generate all TS columns in one excel file which would reduce time for the entire TS creation greatly.

REFERENCES

1. Study Data Technical Conformance Guide. U.S. Department of Health and Human Services, Food and Drug Administration, Center for Drug Evaluation and Research (CDER), Center for Biologics Evaluation and Research (CBER). Version 4.3. March 2019.
2. Study Data Tabulation Model Implementation Guide: Human Clinical Trials. Clinical Data Interchange Standards Consortium (CDISC) Submission Data Standards (SDS) Team. Version 3.3. November 2018.
3. K. Kristin, S. Jerry, W. Fred. 2016. "SDTM Trial Summary Domain: Putting Together the TS Puzzle" Pharmaceutical Industry SAS® Users Group 2016 Conference, DS-11. Berwyn, PA: Accenture Accelerated R&D Services.
4. Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org
5. Django (Version 2.2.1). (2013). Retrieved from https://djangoproject.com.
6. Chandra, R. V., & Varanasi, B. S. (2015). Python requests essentials. Packt Publishing Ltd.
7. Jeri Wieringa, "Intro to Beautiful Soup," The Programming Historian 1 (2012), https://programminghistorian.org/en/lessons/intro-to-beautiful-soup.

CONTACT INFORMATION

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

Jingwei Huang
Pfizer (China) Research & Development Co. Ltd
Ascendas Lotus Business Park, Building 3, 60 Naxian Road, Pudong Zhangjiang Hi-Tech Park
Shanghai, China 201210
Jingwei.Huang@pfizer.com

APPENDIX A, HTML CODES

```
<!DOCTYPE html>
<html lang= "en">
    <head>
        <meta charset= "UTF-8">
        <title>TS Domain Dictionary Crawler</title>
    </head>
    <body>
        <h1>TS Domain Dictionary Crawler</h1>
        <form action="/test/" method="get">
            REGID: Study Number <input type="text" name="StudyNumb"/>
            <input type="submit" value="Search" name='regidsub'><br>
            UNIII:Preferred Substance Name (Such as IBUPROFEN) <input
type="text" name="subname"/>
            <input type="submit" value="Search" name='unisub'><br>
            PCLAS: FDA Established Pharmacologic Class Text
Phrase<input type="text" name="pclasname"/>
            <input type="submit" value="Key word Search"
name='pclassub'><br>
            <a>NDF-RT: National Drug File Reference Terminology Public
Inferred Edition<a/>
            <a>NDF-RT is still under development, please directly visit
link<a/>
            {% if NDFURL %}
                <a href={{ NDFURL}}> here<a/>
            {% endif %}
            <br>
            <a>SNOMED CT: SNOMED Clinical Terms<a/>
            <a>SNOMED CT is also still under development, please
directly visit link<a/>
            {% if SNOMURL %}
                <a href={{ SNOMURL}}> here<a/>
            {% endif %}
            <br>
        </form>

        <table border= "1">
            {% for line in data %}
            <thead>
```

```
    <th>REGID</th>
    <th>Other_IDs Title</th>
    <th>Status</th>
    <th>Study_Title</th>
</thead>
<br>
<tbody>
    <tr>
        <td>{{ line.REGID}} </td>
        <td>{{ line.Other_IDs}} </td>
        <td>{{ line.Status}} </td>
        <td>{{ line.Study_Title}} </td>
    </tr>
</tbody>
{% endfor %}

{% for line in UNDT %}
<thead>
    <th>Substance Name</th>
    <th>UNIII</th>
</thead>
<br>
<tbody>
    <tr>
        <td>{{ line.unisubnam}} </td>
        <td>{{ line.uniid}} </td>
    </tr>
</tbody>
{% endfor %}

{% if PCDT.0.0 %}
<thead>
    <th>Active Moiety Name</th>
    <th>PCLAS Value</th>
</thead><br>
{% endif %}
{% for line in PCDT %}
```

```
                <tbody>
                    <tr>
                        <td>{{ line.0}} </td>
                        <td>{{ line.1}} </td>
                    </tr>
                </tbody>
                {% endfor %}
            </table>
        </body>
    </html>
```

APPENDIX B, BACKEND PYTHON CODE

HTMLREQ is another py file in the same folder which restores the request generic code framework.

```
    from django.shortcuts import render
    from datetime import datetime
    import requests
    from bs4 import BeautifulSoup
    import pandas as pd
    from tabula import read_pdf
    import numpy as np

    from . import htmlreq

    # Create your views here.
    def dict(request):
        AllInfo=[]
        UNIdata=[]
        PCLASDAT=np.array([])
        ndfurl=""
        snomurl=""
        UNIID=[]
        if request.method == "GET":
            if 'regidsub' in request.GET:

                url='https://clinicaltrials.gov/ct2/results'
                recrs=''
                cond=''
                contry=''
```

```
            state=''
            city=''
            dist=''
            term = request.GET.get("StudyNumb", None)
            if term != None and term != "":

kv={'recrs':recrs,'cond':cond,'term':term,'contry':contry,'state':state
,
                    'city':city,'dist':dist}
                r=requests.get(url,params=kv)
                r.raise_for_status()
                r.encoding=r.apparent_encoding
                soup=BeautifulSoup(r.text,"html.parser")

SrchRsltHead=['Status','Study_Title','Conditions','Interventions',
                            'Study
Type','Phase','Sponsor_Collaborators','Funder Type',

'Study_Design','Outcome_Measures','Number_Enrolled','Sex',
                            'Age','REGID','Other_IDs','Title
Acronym',
                            'Study Start','Primary Completion','Study
Completion',
                            'First Posted','Last Updated
Posted','Results First Posted',
                            'Locations','Study Documents','URL']
                SrchRslt=soup.find_all('tr')
                for Rslt_child in SrchRslt:
                    try:
                        InfoColl={}
                        studyInfo=Rslt_child.find_all('td')
                        #InfoColl['Status']=studyInfo[2].text.strip()
                        for i in range(0,23):

InfoColl[SrchRsltHead[i]]=studyInfo[i+2].text.strip()
                        InfoColl["Row"]=studyInfo[0].text.strip()

InfoColl['URL']=studyInfo[3].find('a').attrs['href']
                        AllInfo.append({k:v for k, v in
InfoColl.items() if k in ["REGID","Other_IDs","Status","Study_Title"]})
                    except:
```

```
                        continue


        if 'unisub' in request.GET:
            ULR="https://fdasis.nlm.nih.gov/srs/auto/"
            Substance = request.GET.get("subname", None)
            if Substance != None and Substance != "":
                UNIURL=ULR+Substance
                html=htmlreq.getHTMLText(UNIURL)
                soup=BeautifulSoup(html,"html.parser")
                SrchRslt=soup.find_all('table',attrs={"class","search-
results"})
                for Rslt_child in SrchRslt:
                    UNIID=Rslt_child.find_all('tr')

UNIdata=[{"unisubnam":UNIID[2].text.split(":")[0],"uniid":UNIID[2].text
.split(":")[1]}]


        if 'pclassub' in request.GET:
            URL="https://www.fda.gov/drugs/laws-acts-and-rules/plr-
requirements-prescribing-information"
            html=htmlreq.getHTMLText(URL)
            soup=BeautifulSoup(html,"html.parser")
            SrchRslt=soup.find_all('a')
            for Rslt_child in SrchRslt:
                if "FDA EPC" in Rslt_child.text:

targetURL="https://www.fda.gov"+Rslt_child.attrs['href']
            pclas=requests.get(targetURL)
            with open("test.pdf","wb") as f:
                f.write(pclas.content)
            PCLASDAT=np.array(read_pdf("test.pdf", pages='all',
spreadsheet=True).values.tolist())
            pclaskey = request.GET.get("pclasname", None)
            if pclaskey != None and pclaskey != "":
                tempdat=np.array([x.lower() if isinstance(x, str) else
x for x in PCLASDAT[:,0]])
                rslt=np.char.find(tempdat,pclaskey.lower())
                if len(tempdat[np.where(rslt==0)]) !=0:
                    PCLASDAT=PCLASDAT[np.where(rslt==0)]
```

```
                else:
                    PCLASDAT=np.array([['Key word not found','Please
try again or search blank for all list']])



nci="https://ncit.nci.nih.gov/ncitbrowser/pages/multiple_search.jsf?nav
_type=terminologies"
    ncih=htmlreq.getHTMLText(nci)
    ncisp=BeautifulSoup(ncih,"html.parser")
    ncirlt=ncisp.find_all('a')
    for ncirlt_child in ncirlt:
        if "NDF-RT" in ncirlt_child.text:

ndfurl="https://ncit.nci.nih.gov/"+ncirlt_child.attrs['href']
        if "SNOMED CT" in ncirlt_child.text:

snomurl="https://ncit.nci.nih.gov/"+ncirlt_child.attrs['href']


    return render(request, "homepage.html",
{"data":AllInfo,"UNDT":UNIdata,"PCDT":PCLASDAT,"NDFURL":ndfurl,"SNOMURL
":snomurl})
```