

USING WEB TOOLS TO ENHANCE POPULATION IDENTIFICATION

Carol Martell, UNC Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

In social science research, a study population is often extracted from a large public or limited-access database. The database may not be designed for the purpose of data exploration, but rather tuned, and rightfully so, to carry out some required business task, such as keeping track of licensed drivers. The accurate identification of a study population, exclusively and inclusively, is crucial for the researcher. If manual review is required for case selection, the process of identifying the study population becomes laborious or infeasible. This paper gives an example of using SAS/IntrNet[®] software and HTML to leverage existing data exploration and collection tools into an efficient environment for case-by-case manual review. This customized tool provides, in a single web page, the ability to investigate potential population candidates and build a table of those cases that would algorithmically be left out of the identified population. Some knowledge of SAS/IntrNet software and HTML will be helpful to the audience.

THE GOAL

In this case, the study requires identification of the population of drowsy drivers involved in vehicular crashes during a single year in North Carolina. The crash data form, submitted by law enforcement officials across the state, includes a data item for the driver's physical condition. This variable has two codes of interest: 3 = "fatigue" and 4 = "fell asleep, fainted, loss of consciousness." Another variable contains a narrative description of the crash. A set of drivers could exist whose drowsy state was mentioned in the narrative but whose physical condition is not a code 3 or 4. The goal is to add that set of drivers whose physical condition might have been miscoded or miskeyed, but who could be identified as drowsy by reading the narrative.

THE PROBLEM

The crash data is relational data, having records at both crash and vehicle levels, where there can be multiple drivers per crash. The study will focus on individual drivers (vehicle level), but the narrative field is a crash-level data item. Selecting every driver in a crash having drowsiness mentioned in the narrative would be inappropriate because it is unlikely that every driver is drowsy. Reading the narrative is the only way to identify the correct driver.

The narrative is a free-form text field containing the officer's description of what happened, and usually includes each driver's story. A narrative might contain this text: "V1 stated that he may have fallen asleep. The driver of V2 stated that V1 crossed the centerline before striking his vehicle." If the physical condition for the driver of vehicle one is not 3 or 4, we need to add the driver of vehicle one to the study population.

To identify cases that would be missed and build a table of additions, we need a tool to perform three tasks:

1. Find the crashes with drowsiness mentioned in the narrative and examine the associated driver physical condition codes
2. Manually review the original crash form
3. Compile a list of study additions

We already have the ability to perform these tasks separately. We have permission to view images of the original crash forms using a secure Java web application developed by the North Carolina Department of Motor Vehicles (DMV). Base SAS[®] software can, of course, perform the remaining functions: query and display crash narratives mentioning drowsiness, find and display the associated driver physical condition codes and build a new SAS table of additions. The DMV crash report viewer operates in a browser. We could copy-and-paste our way back and forth from browser window to SAS window, but there would be an awful lot of CTRL-C, ALT-TAB and CTRL-V going on. We need for these tools to interact. We cannot avoid the need for human review of the narratives, but we can streamline the process to keep that human as happy as possible!

THE SOLUTION

SAS/IntrNet tools allow us to access the power of SAS through a web browser. We have a SAS/IntrNet tool that performs narrative searches. A user types a search word into a web form, clicks submit, and all the narratives containing the search word are then displayed in the browser window. For our solution, this htmSQL page needs to be tweaked in the following ways. We need the query to be more complex, looking for more than a single word. The query must also join to another table to find and display the associated driver physical condition codes. To avoid copying and pasting, we would like this query to feed information to the DMV viewer as well as to a data collection tool. We only need to query the narrative data one time to obtain our list of candidates, so we will convert the dynamic page to a static page.

To visually integrate all three tools and to avoid copying and pasting, we use HTML frames to place the DMV Java application along with our SAS/IntrNet tools into a single browser window. The application integrating the required actions is shown in Figure 1. Three adjacent frames perform the three required functions.

The left-hand frame reveals the electronic data values of pertinent fields for every crash where drowsiness was mentioned in the narrative. The right-hand frame provides the means to view the original form as completed by the investigating officer. The center frame is used to collect data, building our SAS table of additional cases.

The HTML code to create these frames looks like this:

```
<frameset cols="30%,30%,40%">
  <frame src="myleftframe.html" name="fl">
  <frame src="mycenterframe.hsql?yr=case=" name="fc">
  <frame src="http://dmvloginpage" name="fr">
</frameset>
```

Three <frame> tags are nested within <frameset> tags. The first contains a static page called "myleftframe.html". The second is a dynamic SAS/IntrNet htmSQL page entitled "mycenterframe.hsql". The third is a call to the DMV Java application login page (none of the actual DMV code is revealed in this paper).

The screenshot displays a web application interface with three frames. The left frame shows crash data for two incidents. The center frame shows a table for corrections already entered. The right frame is the NC DMV Crash Reporting System login page.

Left Frame Data:

Crash Id	Vehicle Number	Correct Physcond Code	Error Type
99003747	vehpos= 1, physcond=1		
99005267	vehpos= 1, physcond= 4		

Center Frame Data:

Crash Id	Vehicle Number	Correct Physcond Code	Error Type

Right Frame Data:

NC DMV Crash Reporting System

Login

User ID:

Password:

Figure 1

THE FRAMES

We examine how each frame functions. Our solution contains a component we cannot modify: the DMV application. Since the other parts must accommodate this application, we will begin there.

THE RIGHT FRAME

When using the DMV application by itself, one proceeds as follows:

Login
Enter the value of a Crash ID into a form and submit
Choose from a list of available images for the crash
View the selected image

The initial filing of a crash report contains information about the crash and repeated sets of information about every vehicle in the crash. The crash narrative appears only once. Supplemental information or forms with corrections to the initial filing may exist. Each filing - initial and subsequent - will have a separate image file and "View" link. When the user clicks "View," a multi-page tiff image is presented in whatever software the machine is configured to use for tiff images. In our case, that is the Windows Picture and Fax Viewer.

We want to skip as far into the DMV application as possible, to save steps in our search for drowsy drivers. We cannot easily skip directly to viewing images, since the number of images is variable. Consequently, our goal is to skip directly to the list of available images for a crash. To do so, we must automate the submission of the Crash ID to the Java application.

When the three-frame HTML page is first opened, the right-hand frame contains the DMV login page. After the required authentication, the DMV application will respond to requests for crash images. We pass the Crash ID to the DMV application from another frame. Whenever the user clicks one of the buttons labeled "View Hardcopy" in the left-hand frame, the Crash ID is submitted to the DMV Java application, and the list of available images is returned to the right-hand frame.

Viewing the form allows us to determine whether the physical condition code was miscoded or miskeyed. When the officer states in the narrative that the driver is drowsy, but does not provide a physical condition code of 3 or 4, we consider it miscoded. If the form actually contains a 3 or 4 for physical condition, but the data does not, we consider it miskeyed.

THE LEFT FRAME

This frame reveals the data values for candidate crashes, facilitates viewing the original crash forms and facilitates data entry for the table of additions. For each candidate crash, the Crash ID is followed by the vehicle number and associated driver physical condition code for every driver in the crash. Physical condition codes that are not 3 or 4 are highlighted in red. Following the physical condition codes we see two buttons that feed information to the other frames, one entitled "View Hardcopy" and the other "Need to Correct." After the buttons, the narrative text for the crash is presented. A horizontal line visually separates each crash from the next.

What we see in the left frame is a static page built using command-line htmSQL. Usually, htmSQL is used via a web browser to dynamically query data and display the results. Every time the page is requested by a browser, the query is executed. We need to run the query once. We can invoke the htmSQL code in command-line mode and direct the query results to a file. Here we see the results directed to *myleftframe.html*.

```
% server-cgi-path/htmSQL myquery.hspl > myleftframe.html
```

We examine the code in *myquery.hspl*. This code must search the narratives to find the crashes of interest, show us the physical condition codes for all the drivers in the crash, give us the two buttons to facilitate viewing original forms and data entry, and show us the narrative text.

The executed query finds the records of interest by searching for a reference to drowsiness in the narrative. The Crash ID, **yr~~case~~**, is numeric. We need both character and numeric versions of the Crash ID for later use in this htmSQL page.

```
{query server="myserver"}  
  
{sql}  
select left(put(yrcase,10.)) as crsh_id, yrcase as yc, *  
  from mylib.mycrashleveldata  
  where nar contains('SLEEP')  
  .  
  .  
  ...other keywords sought...  
  .  
  .
```

```

        or
        nar contains('FATIGUE')
    }/sql}

```

A subquery executes for each row, locating all the vehicle records in the crash by matching with the numeric Crash ID. The value of physical condition is surrounded by tags to show the value as red when the code is not 3 or 4.

```

{eachrow}
    {sql}select vehpos,
        case when physcond not in(3,4)
        then '<font color=red>'||put(physcond,1.)||'</font>'
        else put(physcond,1.)
        end as pc
        from mylib.myvehicleleveledata
        where yrcase={&yc}
    {/sql}

```

Surfacing the data begins. We place the Crash ID on a line by itself.

```

    {&yc}<br>

```

Next, we list each vehicle number and associated driver physical condition, one line per vehicle.

```

    {eachrow}
        vehpos={&vehpos}, physcond={&pc}<br>
    {/eachrow}

```

Two buttons that serve to integrate the three window frames follow. One button is used to send information to the DMV Java server application requesting images of the crash report form(s). The other button supplies the Crash ID to the data collection form in the center frame. Each button is actually a submit button for an HTML form. The target parameter of the form tag in each case directs the result of the form action to appear in the frame having the target name. The center frame has the name **fc** and the right frame has the name **fr**. In each case, the character value for the Crash ID is being passed in a form name/value pair.

The button to view the hardcopy sends the Crash ID to DMV's application:

```

    <form method="put" action="http://DMVactions" target="fr">
    <input type="hidden" name="crashID" value="{&crsh_id}">
    <input type="submit" value="view hardcopy">
    </form>

```

The button facilitating data entry causes the contents of the center frame to be refreshed.

```

    <form method="put" action="mycenterframe.hsql" target="fc">
    <input type="hidden" name="yrcase" value="{&crsh_id}">
    <input type="submit" value="need to correct">
    </form>

```

Last, we show the narrative description for the crash, followed by a horizontal line.

```

    <br>
    {&nar}
    <hr>

```

This completes the processing for a Crash ID and is repeated for every crash selected in the main query.

```

{/eachrow}

{norows}
none found
{/norows}

{/query}

```

The results of executing the code described above appear in the left-hand frame of Figure 1.

THE CENTER FRAME

The data collection takes place in the center frame. This is where we build a SAS table of the drowsy drivers that would have been left out of the study. An HTML form collects the Crash ID, vehicle number and correct physical condition for the drowsy driver. We also include a field indicating whether the officer miscoded the field or the data entry person miskeyed the data.

Since there are several hundred cases to review and it is unrealistic to assume the process will be completed in one sitting, listing the additions already in the table helps the user know where to begin at the next sitting. In *mycenterframe.hspl*, the drowsy drivers already manually identified are shown in an HTML table.

```
{query server="myserver"}  
  Corrections Already Entered<br>
```

First, we provide column headers to label the variables:

```
<table border="1">  
  <tr>  
    <td>Crash Id</td>  
    <td>Vehicle Number</td>  
    <td>Correct Physcond Code</td>  
    <td>Error Type</td>  
  </tr>
```

Next, we query the SAS table of additions:

```
{sql}  
  select yrcase as yc,  
         vehpos as vp,  
         newphyscond as np,  
         errortype as et  
  from mylib.addrowsy  
{/sql}
```

For each observation, we surface the variable values in a table row:

```
{eachrow}  
  <tr>  
    <td>{&yc}</td>  
    <td>{&vp}</td>  
    <td>{&np}</td>  
    <td>{&et}</td>  
  </tr>  
{/eachrow}  
  
</table>
```

Having revealed the existing list of additions, we now provide the data entry form that can add a new row to the table. Here is where we avoid copy-and-paste by placing the value for Crash ID (**yrcase**) into the appropriate form field. The form submits values for the new addition to the SAS/IntrNet Application Dispatcher broker. Since we want the results of running the job to appear in the current frame, we do not need to provide a target parameter to the form tag. The form tag specifies the broker as the action, a hidden input tag provides the program name, and the remaining input tags provide the Crash ID (**yrcase**), vehicle position (**vehpos**), physical condition code (**physcond**), and type of error (**errortype**).

The Crash ID number is automatically entered as the value parameter for an `<input>` tag: `value="{&yrcase}"`. The value is passed to this frame when a user clicks a "Need to Correct" button in the left frame.

```
<form action="/cgi-bin/broker">  
  <input type="hidden" name="_program" value="myproglib.addrowsy.sas">  
  <input type="text" name="yrcase" value="{&yrcase}" size="10"><br>  
  <input type="text" name="vehpos" size="3">vehicle number<br>
```

```

        <input type="text" name="newphyscond" size="3">correct physcond code<br>
        <input type="radio" name="errortype" value="K" checked>miskeyed (entry error)<br>
        <input type="radio" name="errortype" value="C">miscoded (officer error)<br>
        <input type="submit" value="add to corrections">
    </form>
{/query}

```

The form uses the Application Dispatcher component of SAS/IntrNet to execute SAS code. Here is the SAS code in the program "addrowsy.sas," executed when the form is submitted:

```

proc sql;
insert into mylib.addrowsy
    (yrcase, vehpos, newphyscond, errortype)
values
    (&yrcase, &vehpos, &newphyscond, "&errortype");
quit;

```

It is reassuring for the user to see the new row in the table, so we resurface the data. This time we are using Application Dispatcher to write our HTML code, so we must use put statements:

```

data _null_; file _webout;
set mylib.addrowsy;

```

Here we create the HTML table and provide the column headers:

```

if _n_=1 then do;
    put 'Corrections Already Entered<br>';
    put '<table border="1">'
        '<tr>'
            '<td>Crash Id</td>'
            '<td>Vehicle Number</td>'
            '<td>Correct Physcond Code</td>'
            '<td>Error Type</td>'
        '</tr>' ;
end;

```

Next, we write a row for every observation:

```

Put    '<tr><td>'
        yrcase
        '</td><td>'
        vehpos
        '</td><td>'
        newphyscond
        '</td><td>'
        errortype
        '</td></tr>';
run;

```

Now we must close our HTML table:

```

data _null_;
file _webout;
put '</table>';
run;

```

USING OUR SOLUTION

We examine the first crash narrative on the left: "V1 STATED THAT HE FELL ASLEEP..." which clearly indicates a driver fell asleep. There appears to be only one driver in the crash, and the physical condition code is neither 3 nor 4 (the red color draws our attention to that fact). We want to add the case to our study file. For verification, we wish to view the original crash report(s).

Before we may view any hardcopies, we must authenticate to the DMV system in the right-hand frame. After providing our user ID and password to DMV, we click “View Hardcopy” in the left-hand frame, which sends a request to DMV and returns a list of images we may view in the right-hand frame (Figure 2). We click “View” in the right-hand frame to request the image files, which open in Windows Picture and Fax Viewer (Figure 3). There, we see that the officer had coded the physical condition as 4. Since the data reveals a 1, we consider this to be a data entry error.

We click “Need to Correct” in the left-hand frame, which places the Crash ID into the data collection form in the center frame (Figure 4). Next, we type in the vehicle number and the physical condition code of 4. We choose “miskeyed (entry error)” and click “Add to Corrections”(Figure 5). The vehicle data is inserted into our table of additions, and is now listed under “Corrections Already Entered” (Figure 6).

Figure 2

Figure 3

99003747

vehpos= 1, physcond=1

[view hardcopy](#)

[need to correct](#)

V1 STATED THAT HE FELL ASLEEP H
IS VEH LEFT THE RDWY AND WENT
DOWN A VERY STEEP
EMBANKMENT :

99005267

vehpos= 1, physcond= 4

[view hardcopy](#)

[need to correct](#)

1 TRAV E ON RP1363 RNAOFF IRIGHT
FEL ASLEEP :

Corrections Already Entered

Crash Id	Vehicle Number	Correct Physcond Code	Error Type
99003747			
	vehicle number		
	correct physcond code		
	☒ miskeyed (entry error)		
	☐ miscoded (officer error)		
	add to corrections		

NC DMV Crash Reporting System

External User Request

Need [Help](#) with this screen?

Enter Crash ID: 99003747

[Submit](#)

Documents available for crash ID 99003747 :

Initial [View](#)
Filing

Figure 4

99003747

vehpos= 1, physcond=1

[view hardcopy](#)

[need to correct](#)

V1 STATED THAT HE FELL ASLEEP H
IS VEH LEFT THE RDWY AND WENT
DOWN A VERY STEEP
EMBANKMENT :

Corrections Already Entered

Crash Id	Vehicle Number	Correct Physcond Code	Error Type
99003747			
1	vehicle number		
4	correct physcond code		
	☒ miskeyed (entry error)		
	☐ miscoded (officer error)		
	add to corrections		

NC DMV Crash Reporting System

External User Request

Need [Help](#) with this screen?

Enter Crash ID: 99003747

Figure 5

99003747

vehpos= 1, physcond=1

[view hardcopy](#)

[need to correct](#)

V1 STATED THAT HE FELL ASLEEP H
IS VEH LEFT THE RDWY AND WENT
DOWN A VERY STEEP
EMBANKMENT :

Corrections Already Entered

Crash Id	Vehicle Number	Correct Physcond Code	Error Type
99003747	1	4	R

NC DMV Crash Reporting System

External User Request

Need [Help](#) with this screen?

Enter Crash ID: 99003747

Figure 6

CONCLUSION

We have seen how it is possible to combine existing tools into a single, interactive tool. With limited HTML knowledge, some SAS/IntrNet skills, and a little imagination, we can make feasible that which might otherwise be left undone.

ACKNOWLEDGEMENTS

We wish to thank Rosa Gill, retired Director of Traffic Records, NC Department of Motor Vehicles, for her many years of service and for granting Highway Safety Research Center access to the Crash Reporting System for our research purposes.

CONTACT INFORMATION

Your comments and questions are welcome and valued: Contact the author at:

Carol Martell
UNC Highway Safety Research Center
730 Martin Luther King, Jr. Blvd
Chapel Hill, NC 27514
Work Phone: (919) 962-8713
Fax: (919) 962-8710
Email: carol_martell@unc.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.