One Click to Create a List of All MEPS-HC ZIP Files' Download URLs: Web Scraping with SAS® and Python

Pradip K. Muhuri, Agency for Healthcare Research and Quality;
Charles Z.X. Han, Ryman Healthcare Ltd.;
John Vickery, North Carolina State University

ABSTRACT

This paper presents a case study of web scrapping the Medical Expenditure Panel Survey-Household Component (MEPS-HC) sites with SAS® and Python. Each program dynamically constructs an up-to-date list of over one thousand clickable MEPS-HC ZIP files' download links (Uniform Resource Locators or URLs) and several identifiers and then saves the list into an Excel file. Once you automate the creation of a list of all those URLs in an Excel file using one of these programs (SAS or Python), you can directly initiate ZIP file downloads by simply clicking the relevant URLs from that file outside of the Internet, an easy alternative to navigating multiple pertinent websites for the same task. These programs are reusable to obtain an updated list of MEPS-HC file download links as more data sets are added to the website annually from the ongoing panel survey.

INTRODUCTION

The Agency for Healthcare Research and Quality (AHRQ) has made various data files available from the ongoing Medical Expenditure Panel Survey-Household Component (MEPS-HC[1]) to the public on its website. On this site (https://meps.ahrq.gov/data_stats/download_data_files.jsp), you can manually select an individual data year or all available years from a dropdown list and check the box that represents the file type(s)[2] for your search and then navigate through the site for the desired public-use file (PUF). The PUF site includes links for the data file documentation, survey questionnaires, codebooks, SAS®/Stata/R programs to read the raw data into the individual file format, and ZIP files' download URLs.

Over one thousand ZIP files' download URLs for ASCII, Excel, SAS transport, SAS V9, and Stata files exist across all MEPS-HC PUF websites. However, a consolidated list of all those URLs is unavailable at a single location. Therefore, finding an HTML link to trigger a ZIP file download can be tedious and time-consuming when desiring many such files because the process requires manual navigation across multiple linked websites.[3] The inefficiency increases with the number of file downloads. The goal of web scraping the MEPS-HC sites with SAS® and Python here is to dynamically construct an up-to-date list of zip files' download URLs and several identifiers and save the list in individual Excel files for easy use.

[1] MEPS is a set of large-scale on-going surveys of families and individuals, their medical providers, and employers across the United States. It is the complete data source on the cost and use of health care and health insurance coverage for the U.S. civilian noninstitutionalized population [1]. See https://www.meps.ahrq.gov/mepsweb/about_meps/survey_back.jsp for a complete description of MEPS, including its Household Component (HC).

[2] Examples of file types include Household Full Year File, Household Event File (e.g., office-based medical provider visits file, outpatient visits file, emergency room visits file, hospital in-patient stays file, home health visits file, dental visits file, and prescribed medicines file), and Pooled Linkage File.

[3] For the data year 2023, the schedules for public-use file releases spread over ten months from March through December (https://meps.ahrq.gov/mepsweb/about_meps/releaseschedule.jsp).

SAS® PROGRAM

We present the SAS program[4] (shown in Appendix I[5]) in six steps below.

Step 1: Fetch the content of the main MEPS-HC web page

Line 3: The FILENAME statement associates the device (TEMP) to the file reference (fileref for short) SOURCE. The TEMP device allows us to create a temporary file that exists only as long as the filename is assigned.

Lines 4-7: The code performs the HTTP requests to the specified URL. It writes the response body into the fileref SOURCE.

Step 2: Extract data year values by parsing HTML content

Line 11: The DATA statement creates an out**put** SAS data set YEAR_VALUES.

Line 14 The INFILE statement specifies the fileref SOURCE and other relevant options (e.g., DLM='<', EXPANDTABS, and TRUNCOVER).

The following are sample data records of the PROC HTTP response.

```
<td width="523" height="0"><span class="contentStyle">
            <select id="datayear" size=1 name="cboDataYear">
              <option value="All">All available years</option>
              <option value="2021">2021</option>
              <option value="2020">2020</option>
              <option value="1996">1996</option>
              <option value="Projected Years">Projected Years</option>
```

Line 15: We use the @'*character-string*' syntax[6], that is, two @ column pointers, to move the input pointer to the correct location in the data record to read the value of the variable YEAR (specified first in the LENGTH statement).

The ?? modifier after YEAR suppresses the invalid data message and prevents the automatic variable _ERROR_ from being set to 1, which results from reading invalid data for the numeric variable. In that case, the variable YEAR will get a missing value.

Line 16: If the expression is true, SAS will include the current observation in the output data set created. Thus the DATA step creates a data set YEAR_VALUES that contains only those observations with non-missing values for the variable YEAR.

[4] We used SAS® 9.04.01M6 X64_10PRO in the Windowing environment.
[5] One can use this online tool https://remove-line-numbers.ruurtjan.com/ to remove line numbers from the SAS program in the APPENDIX I before running it.
[6] See the @'character-string' section https://go.documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lestmtsref/n0oaql83drile0n141pdacojq97s.htm#p10kxp5ghaqig7n1qcmwle2swfox.

```
NOTE: 3268 records were read from the infile (system-specific pathname).
      The minimum record length was 0.
      The maximum record length was 452.
NOTE: The data set WORK.YEAR_VALUES has 26 observations and 1 variables.
```

Line 19: The FILENAME statement with the CLEAR option disassociates the fileref from the file.

Step 3: Call PROC HTTP and extract MEPS public-use file numbers, file names, and data year via macro processing

Lines 22-78: The %MACRO statement defines the macro program giving it a name (GET_YEAR) and specifying its only parameter, YEAR, and the %MEND statement closes the macro, wrapping the following in the macro definition:

- %LET statements to create macro variables by assigning different URL parts to them and then concatenating those macro variables to another macro variable to construct the year URL.
- PROC HTTP to retrieve an HTTP response body for further processing in a DATA step to construct hundreds of MEPS public-use file numbers, MEPS file names, and data years.

Lines 81-84: The code invokes the macro by enclosing the macro call as an argument to the CALL EXECUTE in a DATA _NULL_ step. The macro call %GET_YEAR generates code for 26 PROC HTTP and 26 DATA steps, adding them to the program stack for further processing. As examples, the resolved values of two macro variables from the first iteration are below.

- &YEAR resolves to 1996.
- &YEAR_URL resolves to https://meps.ahrq.gov/data_stats/download_data_files_results.jsp?cboDataYear=1996&buttonYearandDataType=Search.

Lines 35-38: The macro-generated code performs the HTTP requests to the specified URL. It writes the response body into the fileref YEARRESP.

Line 40: The DATA statement creates a data set (e.g., YEAR_LIST_2021) with a KEEP= option and puts it in the temporary WORK library.

Line 41: The LENGTH statement specifies the length of variables PUF_NUM, MEPS_FILE, and DATA_YEAR created subsequently

Line 42: The INFILE statement specifies the file reference name YEARRESP associated with the external file, and LENGTH=, LRECL=, and END= options.

Lines 45-54: As a good programming practice, we combine the RETAIN statement and 'IF _N_ = 1 THEN …' with the PRXPARSE function to create a regular expression only once rather than for each DATA step iteration.

The regular expression in the argument to the PRXPARSE function uses a pair of slashes "/…/" as the delimiters. Due to their special meanings in regular expressions, some characters, as shown below, with a backslash (\, known as an escape symbol) match them literally in the source string.

- \. matches the literal period.
- \? matches the literal question mark.
- \/ matches the literal forward slash.

The following metacharacters perform particular actions when searching a source string.
- .+ matches any single character one or more times.
- (.+) represents the capture buffer due to parentheses (around the .+), instructing SAS to extract and/or replace specific portions of the source string.

PRXPARSE passes and compiles the regular expression, returning three numbered pattern identifiers (i.e., PRX_PUF, PRX_MEPS, and PRX_DATA_YEAR) manipulated subsequently using the specific CALL routine and other PRX and SAS functions to create three variables (PUF_NUM, MEPS_FILE, and DATA_YEAR) for file public use file numbers (e.g., HC-230), MEPS files names (e.g., 2021 Food Security File) and data years (e.g., 2021).

Line 57-58: Within the DO WHILE(NOT EOF) END code block, the INPUT statement includes a character variable named HTML_LINE to read values of the raw data field with the $VARYING32767. informat. It also specifies a required numeric variable, RECLEN**,** due to LENGTH = RECLN in the INFILE statement. The DO WHILE loop will continue as long as the NOT EOF condition is true (i.e., the end of the data file has not been reached).

Lines 59-72: Within the same loop, when PRXMATCH finds a position of the matched value in the source string, CALL PRXPOSN uses the PRXPARSE results to find the start position (START) and length (END) of the numbered capture buffer within the string. These values are fed into the SUBSTR function to extract values for the PUF number (PUF_NUM), the MEPS file name (MEPS_FILE, code example below from APPENDIX I), and the data year (DATA_YEAR), respectively.

```
meps_file = substr(html_line, start, end);
```

Example output: `2020 Full Year Population Characteristics File<br/>(HC-219 replaced by HC-224)`

Line 66: The PRXCHANGE function, as shown below from APPENDIX I, removes the string `<br/>` from the value of the variable MEPS_FILE.

```
meps_file = prxchange('s/<.+>//', -1, meps_file);
```

In the argument to the PRXCHANGE function,
- 's' before the first delimiter (/) indicates that the first Perl regex should be substituted.
- '.+' matches any character that occurs one or more times to find the text to be replaced
- // specifies that the replacement string is blank.
- -1 indicates that the function will search and substitute until the end of the source is reached.
- MEPS_FILE is the source string to be searched.

Example output: `2020 Full Year Population Characteristics File(HC-219 replaced by HC-224)`

Line 71: Finally, the OUTPUT statement writes the current observation to the SAS data set defined in the DATA statement.

Example-SAS Log (partial) from the processing of the HTTP response body of https://mep.ahrq.gov/data_stats/download_data_files_results.jsp?cboDataYear=2021&buttonYearandDataType=Search

```
NOTE: The infile YEARRESP is:
      Filename=c:\Data\hc_response.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=60915,
      Last Modified=15Aug2023:18:45:11,
      Create Time=15Aug2023:18:37:53

NOTE: DATA STEP stopped due to looping.
NOTE: 1146 records were read from the infile YEARRESP.
      The minimum record length was 0.
      The maximum record length was 452.
NOTE: The data set WORK.YEAR_LIST_2021 has 14 observations and 3 variables.
```

Of the various SAS data sets created, the PROC PRINT output from WORK.YEAR_LIST_2021 is below (code not shown in the program in APPENDIX I, but executed separately).

```
puf_num      meps_file                                                    data_year

NHIS Link    1996 - 2021 MEPS-NHIS Link Files                             1996-2021
HC-230       2021 Food Security File                                      2021
HC-229H      2021 Home Health File                                        2021
HC-229G      2021 Office-Based Medical Provider Visits File               2021
HC-229F      2021 Outpatient Visits File                                  2021
HC-229E      2021 Emergency Room Visits File                              2021
HC-229D      2021 Hospital Inpatient Stays File                           2021
HC-229C      2021 Other Medical Expenses File                             2021
HC-229B      2021 Dental Visits File                                      2021
HC-229A      2021 Prescribed Medicines                                    2021
HC-228       2021 Full Year Population Characteristics File               2021
HC-227       2021 Jobs File                                               2021
HC-036BRR    MEPS 1996-2021 Replicate File for BRR Variance Estimation    2021
HC-036       MEPS 1996-2021 Pooled Linkage File for Common Variance Structure    2021
```

Lines 88-91: The DATA step concatenates the SAS data sets (e.g., YEAR_LIST_1996, YEAR_LIST_1997, … YEAR_LIST_2021) using the name prefix (YEAR_LIST) followed by a colon(**:**) to select those ending in numeric suffixes (i.e., YEAR values) in the SET statement.

Lines 94-96: The NODUPKEY option in the PROC SORT statement creates an output data set YEAR_VALUES with no duplicate observations (i.e., only one observation for the _ALL_ variable values).

The partial SAS Log (some notes are skipped)[7]:

```
NOTE: The data set WORK.YEAR_LIST has 515 observations and 3 variables.
NOTE: 79 observations with duplicate key values were deleted.
NOTE: The data set WORK.YEAR_LIST has 436 observations and 3 variables.
```

[7] The duplicate values include two or more occuerences of the MEPS Longitudinal Data File from various panels, the 2000-2013 Employment Variables File, the 2002-2009 Risk Adjustment Scores File, the 1996-2001 Risk Adjustment Scores File (HC-081 replaced by HC-092), the 2001 and 2002 MEPS HC Survey Data (CD-ROM), the Multum

Step 4: Call PROC HTTP, and extract public-use file (PUF) formats and ZIP files' download URLs via macro processing

Lines 99-145: The %MACRO statement defines the macro program giving it a name (GET_PUF) and specifying its only parameter, PUFNUM, and the %MEND statement closes the macro, wrapping the following in the macro definition:

- %LET statements to create two macro variables assigning different URL parts to them and then concatenating those macro variables to another macro variable to construct the MEPS PUF URLs

- PROC HTTP to retrieve an HTTP response body for further processing in a DATA step.

Lines 148-151: As before, the code invokes the macro by enclosing the macro call as an argument to the CALL EXECUTE in a DATA _NULL_ step. Here, the macro call %GET_PUF generates code for hundreds of PROC HTTP and DATA steps, adding them to the program stack to construct MEPS public-use file numbers, file formats, and ZIP files' download URLs.

As examples, the resolved values of the macro variables from the first iteration are below.

- &PUFNUM resolves to HC-092.
- &PUF_URL resolves to https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC092.
- PUF_LIST_%sysfunc(translate(&PUFNUM, '_', '-')) resolves to PUF_LIST_HC_092.

Lines 109-112: The code performs the HTTP requests to the specified URL. It writes the response body into the fileref PUFRESP.

Line 114: The DATA statement creates a data set (e.g., PUF_LIST_HC_092) with a KEEP= option and puts it in the temporary WORK library.

Line 115: The LENGTH statement specifies the length of variables PUF_NUM, FILE_FORMAT, and ZIP_LINK created subsequently.

Line 116: The INFILE statement specifies the file reference name PUFRESP associated with the external file and LENGTH=, LRECL= option, and END= options.

Lines 119-124: As before, PRXPARSE passes and compiles the regular expression, returning two numbered pattern identifiers (i.e., PRX_DATA_FILE and PRX_ZIP) manipulated subsequently CALL routine, and other PRX and SAS functions to create variables two variables (DATA_FILE and ZIP_LINK) for file formats (e.g., `ASCII format`) and ZIP file links (e.g., https://meps.ahrq.gov/data_files/pufs/h228/h228dat.zip).

The usage of the metacharacters ('.+' and '(.+)') in the regular expression are discussed earlier. The metacharacter, '*' , matches 0 or more occurrences of the preceding element.

Lexicon Addendum Files to MEPS Prescribed Medicines Files 1996-2013, and the 1999 and 2000 MEPS HC Survey Data (CD-ROM).

Lines 126-128: Within the DO WHILE(NOT EOF) END code block, the INPUT statement includes a character variable named HTML_LINE to read values of the raw data field with the $VARYING32767. informat. It also specifies a required numeric variable, RECLEN, due to LENGTH = RECLN in the INFILE statement. Then comes the macro variable &pufnum, whose value is assigned to the DATA step character variable PUF_NUM. The DO WHILE loop will continue as long as the NOT EOF condition is true (i.e., the end of the data file has not been reached).

Lines 129-137: Within the same loop, when PRXMATCH finds a position of the matched value in the source string, CALL PRXPOSN uses the results from the PRXPARSE to find the start position (START) and length (END) of the numbered capture buffer within the string. These values are fed into the SUBSTR function to extract values for the file format (FILE_FORMAT). The CATX and SUBSTR functions are used to create the ZIP file link (ZIP_LINK).

Line 138: Finally, the OUTPUT statement writes the current observation to the SAS data set defined in the DATA statement.

Example-SAS Log (partial) from the processing of the HTTP response body of
https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-228

```
NOTE: DATA STEP stopped due to looping.
NOTE: 1170 records were read from the infile PUFRESP.
      The minimum record length was 0.
      The maximum record length was 1338.
NOTE: The data set WORK.PUF_LIST_HC_228 has 5 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.09 seconds
      cpu time            0.09 seconds
```

Line 143: The FILENAME statement with the CLEAR option disassociates the fileref from the file.

Lines 154-156: The DATA step concatenates the SAS data sets[8] using the name prefix (PUF_LIST) followed by a colon(**:**) to select those ending PUF numbers as suffixes. The resulting output data set is WORK.PUF_LIST.

The next is the PROC PRINT output from WORK.PUF_LIST_228 (code not shown in the program in APPENDIX I, but executed separately).

```
PUF_num    file_format                                        zip_link

HC-228     ASCII format            https://meps.ahrq.gov/data_files/pufs/h228/h228dat.zip
HC-228     SAS transport format    https://meps.ahrq.gov/data_files/pufs/h228/h228ssp.zip
HC-228     SAS V9 format           https://meps.ahrq.gov/data_files/pufs/h228/h228v9.zip
HC-228     Stata format            https://meps.ahrq.gov/data_files/pufs/h228/h228dta.zip
HC-228     XLSX format             https://meps.ahrq.gov/data_files/pufs/h228/h228xlsx.zip
```

[8] Twenty-three data sets (out of 436 concatenated data sets to form a single data set WORK.PUF_LIST using the SET statement) have missing values on the ZIP_LINK variable. These 23 public use files were ad-hoc files and were later updated with new PUF numbers and data download URLs on the website. Thus old data download URLs no longer exits on the website and thus ZIP_LINK values are missing. Here are examples of old and repalced URL, respectively: https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-002 (HC-012 replaced HC-002).
https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-012.

Lines 159-161: The NODUPKEY option in the PROC SORT statement creates an output data set PUF_LIST with no duplicate observations (i.e., only one observation for the _ALL_ variable values).

The partial SAS Log:

```
NOTE: 0 observations with duplicate key values were deleted.
NOTE: The data set WORK.PUF_LIST has 1209 observations and 3 variables.
```

Step 5: One-to-many matching

Lines 164-171: The PROC SQL performs a one-to-many match with the tables YEAR_LIST and PUF_LIST based on the key column PUF_NUM and creates a table MEPS_ZIP_LINKS. Note the following.

- The table YEAR_LIST has a unique value for the column PUF_NUM.
- The table PUF_LIST includes multiple rows with the same value as the column PUF_NUM.
- The output table MEPS_ZIP_LINKS has a unique value for the column ZIP_LINK.

The SAS Log:

```
NOTE: Table WORK.MEPS_ZIP_LINKS created, with 1209 rows and 5 columns.
```

Step 6: Create listings with PROC REPORT and output them into an Excel spreadsheet

Lines 179-194: The REPORT procedure identifies the final merged data set, displaying the values of all specified variables. The CALL DEFINE associates the URL with ZIP_LINK. The ODS EXCEL statement directs the PROC REPORT output to a file listed in the FILE= option. With the execution of the ODS EXCEL CLOSE; statement, the Excel file gets created (part of the output from the Excel spreadsheet shown in APPENDIX II).

The partial SAS Log:

```
NOTE: Writing EXCEL file: c:\SESUG_2023\SAS_Solution_WS_2023-08-08.xlsx
```

PYTHON PROGRAM

We used Python (Version 3.9.12) in the Spyder IDE (Version 5.1.5) and JupyterLab Python Notebook (Version 6.4.5) as our second solution for web scraping. We present the Python program (as shown in APPENDIX III[9]) in four steps below.

Step 1: Import libraries

Lines 8-11: The program imports necessary libraries, including Requests and BeautifulSoup[10] to scrape and parse MEPS-HC websites and dynamically create a list of over one thousand URLs that would trigger data downloads.

[9] One can use this online tool https://remove-line-numbers.ruurtjan.com/ to remove line numbers from the Python program in the APPENDIX III before running it.

[10] BeautifulSoup, Scrapy, and Selenium are the three main Python-based tools commonly used for web scraping.

Step 2: Get the list of numerical year options and saving as the list of tuples with year and URL

Lines 18-19: This code uses the 'requests' library to send a GET request to a website with the URL "https://meps.ahrq.gov/data_stats/download_data_files.jsp". The response from the website is then passed to BeautifulSoup, another library, which parses the HTML text into an object called main_soup.

Line 23: The code defines a base URL as "https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.jsp?cboDataYear=", used to construct the URLs for downloading data files.

Lines 24-26: The 'year_url_suffix' variable is set to "&buttonYearandDataType=Search", which will be appended to each year-specific URL. Next, the code extracts the available years for data download from 'main_soup'. It does so by finding the first 'select' element with the name attribute set to "cboDataYear", which is used to choose a specific year for data download. The '.select("option")' method is then called on this `select` element, which returns all the 'option' elements. The 'year_options' variable now contains a list of all the available years for data download.

The code below prints the list `year_options`.

```
print(year_options)
```

The output:

```
[<option value="All">All available years</option>, <option value="2021">2021</option>, <option value="2020">2020</option>, <option value=
"2019">2019</option>, <option value="2018">2018</option>, <option value="2017">2017</option>, <option value="2016">2016</option>, <
option value="2015">2015</option>, <option value="2014">2014</option>, <option value="2013">2013</option>, <option value="2012">201
2</option>, <option value="2011">2011</option>, <option value="2010">2010</option>, <option value="2009">2009</option>, <option value
="2008">2008</option>, <option value="2007">2007</option>, <option value="2006">2006</option>, <option value="2005">2005</option>,
<option value="2004">2004</option>, <option value="2003">2003</option>, <option value="2002">2002</option>, <option value="2001">20
01</option>, <option value="2000">2000</option>, <option value="1999">1999</option>, <option value="1998">1998</option>, <option valu
e="1997">1997</option>, <option value="1996">1996</option>, <option value="Projected Years">Projected Years</option>]
```

Lines 28-32: The code creates a 'year_url_list' by iterating through each "option" element in 'year_options'. If the "value" attribute of the "option" element is a digit, a tuple is added to 'year_url_list'. The first element of the tuple is the value of the "value" attribute, representing the year for data download. The second element is a URL constructed by concatenating the base URL with the year value, passed through the 'quote()' method to ensure it is appropriately encoded for use in a URL. Finally, the 'year_url_suffix' variable is appended to the URL.

Overall, this code retrieves available data years associated with the public-use file name from the website, constructs URLs for downloading data files for each year, and stores them in a list of tuples called 'year_url_list'.

Below is the additional code (not shown in the Python program in Appendix III) to print the first two elements in the list `year_url_list`.

```
for item in year_url_list[:2]:   print(item)
```

BeautifulSoup() of the Python bs4 library enables one to extract data (specific elements) from a single webpage at a time effectively. Scrapy is a web scraping framework that can crawl various web pages, downloading, parsing, and storing data, whereas Selenium can automate navigating to sites and scrape the webpage content dynamically.

The output:

```
('2021', 'https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.j
sp?cboDataYear=2021&buttonYearandDataType=Search')

('2020', 'https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.j
sp?cboDataYear=2020&buttonYearandDataType=Search')
```

Step 3: Get the list (i.e., year_url_list) of tuples to obtain "HC-" zip file links

Lines 38-39: The `base_hc_url` variable remains the same, representing the base URL for constructing Zip files' download URLs. The `zip_link_list` is an empty list that will store dictionaries containing information about the ZIP file download links.

Lines 40-43: The code then enters a `for` loop that iterates through each tuple in `year_url_list`. Inside the loop, it sends a GET request to the year-specific URL, and if the request is successful (`response.raise_for_status()`), it proceeds to extract the data using `pd.read_html()`.

Lines 45-46: The `pd.read_html()` function parses the HTML response text and attempts to extract tables from it. Each table is represented as a DataFrame in the `dfs` list. Next, the code initializes an empty DataFrame called `hc_df` to store the relevant data from the tables.

Lines 47-53: For each DataFrame `df` in `dfs`, the code checks if it contains the column "File(s), Documentation & Codebooks". If it does, the code concatenates `df` with `hc_df`, drops rows with missing values in the "PUF no." column, selects only rows where the "PUF no." contains "HC-", drops duplicate rows based on the "PUF no.", and appends the MEPS PUF URL by combining `base_hc_url` with the values in the "PUF no." column.

Lines 54-66: If `hc_df` is not empty, the code renames the columns of `hc_df` and iterates over each row using `hc_df.itertuples()`. For each row, it sends a GET request to the MCH-HC data file URL (`row.hc_url`), raises an exception if the request fails and parses the HTML response using `BeautifulSoup`.

Lines 68-83: If successful, the code extracts the name of the MEPS-HC PUF file from the HTML using `hc_soup.find(class_="OrangeBox").text` and searches for a specific table cell ("td") containing the text "Data File". If found, it creates a dictionary called `zip_link_dict` and populates it with information such as the PUF number, data year, PUF name, file format, and the ZIP file download link. The `zip_link_dict` is then appended to `zip_link_list`.

The code below prints selected elements from `zip_link_list`.

```
    for item in zip_link_list[:2]:   print(item)
```

The output:

```
{'PUF_num': 'HC-228', 'data_year': '2021', 'meps_file': 'MEPS HC-228: 2021 Full Year Popul ation
Characteristics File', 'file_format': 'Data File, ASCII format', 'zip_link': 'https:
//meps.ahrq.gov/data_files/pufs/h228/h228dat.zip'}

 {'PUF_num': 'HC-228', 'data_year': '2021', 'meps_file': 'MEPS HC-228: 2021 Full Year Popul ation
Characteristics File', 'file_format': 'Data File, SAS transport format', 'zip_link':

 'https://meps.ahrq.gov/data_files/pufs/h228/h228ssp.zip'}
```

Lines 84-90: Any exceptions that occur during the execution of the code are caught using `try`/`except` blocks, and error messages are printed.

Overall, this code retrieves ZIP files' download URLs for each available year based on the information obtained from the previous code and appends the relevant information to the `zip_link_list`.

Step 4: Create an Excel file from the Pandas DataFrame

Line 100: The code first creates the DataFrame `meps_df` using the `pd.DataFrame()` constructor and passing `zip_link_list` as an argument, which contains the information about the ZIP file download links.

Line 101: The code removes duplicate rows in `meps_df` using the `drop_duplicates()` method with the `inplace=True` argument to modify the DataFrame in-place.

Line 102: The code modifies the "meps_file" column in `meps_df` by splitting the values using `str.split(", ", n=1)` and selecting the second part of the split using `str[-1]`. It extracts the MEPS file name and updates the values in the "meps_file" column.

Line 103: The code modifies the "file_format" column in `meps_df` by splitting the values using `str.split(", ", n=1)` and selecting the second part of the split using `str[-1]`. It extracts the file format and updates the values in the "file_format" column.

Line 104: The code performs multi-key sorting using the 'sort_values' attribute with the ascending= argument using True or False for the same number of values.

Line 105: The code prints the first few rows of the modified DataFrame using the `head()` method.

Line 111: The variable `today` represents the current date. It uses `pd.Timestamp("now")` to obtain the current timestamp and `strftime("%Y-%m-%d")` to format it as "YYYY-MM-DD".

Line 112: The `to_excel()` method is used to write the DataFrame to an Excel file, with the `index=False` argument to exclude the DataFrame index from the output. The code saves the DataFrame to an Excel file with a dynamically generated filename. It uses f-string formatting to include the `today` variable in the filename, resulting in a filename like "C:\SESUG_2013\Python_Solution_WS_YYYY-MM-DD.xlsx".

Overall, this code modifies and displays the DataFrame `meps_df`, generates the current date, and saves the DataFrame to an Excel file with a filename that includes the current date in the "output" folder.

Comparing SAS and Python Output

We read SAS and Python-generated Excel files using the read_excel() pandas method into pandas DataFrames. We then compared the two DataFrames using pandas compare(). This comparison confirms that SAS and Python applications-generated file download URL lists are identical.

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='openpyxl')
df_sas = pd.read_excel(r"C:\SESUG_2023\SAS_Solution_WS_2023-08-12.xlsx")
df_py = pd.read_excel(r"C:\SESUG_2023\Python_Solution_WS_2023-08-12.xlsx")
diff = df_sas.compare(df_py, keep_equal=True, keep_shape = True)
print(diff)
```

The partial output:

```
    data_year  ...                                  zip_link
         self  ...                                     other
0        2021  ...  https://meps.ahrq.gov/data_files/pufs/h227/h22...
1        2021  ...  https://meps.ahrq.gov/data_files/pufs/h227/h22...
2        2021  ...  https://meps.ahrq.gov/data_files/pufs/h227/h22...
3        2021  ...  https://meps.ahrq.gov/data_files/pufs/h227/h22...
4        2021  ...  https://meps.ahrq.gov/data_files/pufs/h227/h22...
      ...  ...                                               ...
1204     1996  ...    https://meps.ahrq.gov/data_files/pufs/h17ssp.zip
1205     1996  ...    https://meps.ahrq.gov/data_files/pufs/h24dat.zip
1206     1996  ...    https://meps.ahrq.gov/data_files/pufs/h24ssp.zip
1207     1996  ...    https://meps.ahrq.gov/data_files/pufs/h41dat.zip
1208     1996  ...    https://meps.ahrq.gov/data_files/pufs/h41ssp.zip

[1209 rows x 10 columns]
```

CONCLUSION

The paper offers SAS® and Python solutions to dynamically create a list of MEPS-HC files' download URLs representing various formats (e.g., ASCII, Excel, SAS transport, SAS V9, and Stata files). Our comparison of the results confirms that SAS and Python applications-generated file download URL lists are identical. This work found that code efficiencies varied between the two programs. For example, Python's libraries BeautifulSoup and urllib.request for HTTP Requests came in handy to perform web scraping. On the other hand, in performing the same task, some users may find the SAS program much more intuitive. However, we see the Python solution as the most efficient due to its flexibility, ease of use, and minimal coding.

These programs are reusable to obtain an updated list of MEPS-HC ZIP files' download links as more data sets are added to the website annually from the ongoing panel survey. In other words, future use of our programs will likely result in more such URLs depending on the timing of their execution, as the target web page and the linked websites get updated with new files. Once you automate the creation of a list of all MEPS-HC ZIP files' download URLs in an Excel file using one of these programs (SAS or Python), you can directly initiate ZIP file downloads by simply clicking the relevant URLs from that file outside of the Internet, an easy alternative to navigating multiple pertinent websites for the same task.

REFERENCES

1. Cohen, J., S. Cohen, and J. Banthin. 2009. "The Medical Expenditure Panel Survey: A National Information Resource to Support Health Care Cost Research and Inform Policy and Practice." *Medical Care*, 47:S44-50.

ACKNOWLEDGMENTS

The views expressed in this paper are those of the authors, and no official endorsement by the Health and Human Services or the Agency for Healthcare Research and Quality (AHRQ) is intended or should be inferred. However, the authors thank AHRQ staff and Kurt Bremser in SAS Support Communities for reviewing earlier manuscript versions and providing valuable comments. Finally, the authors are responsible for any errors or omissions.

Contribution Statements: Pradip K. Muhuri conceived the idea of the paper, drafted it, and wrote the earlier version of the SAS and Python programs in it. Charles Z.X. Han subsequently revised the Python program and contributed to the code description. Later, John Vickery optimized the SAS and Python programs to their current form. Finally, Pradip K. Muhuri worked on another round of manuscript revisions, to which Charles Z.X and John Vickery contributed.

CONTACT INFORMATION

Your comments are valued and encouraged. Contact the authors at:

Pradip K. Muhuri, PhD
AHRQ/CFACT
5600 Fishers Lane
Rockville, MD 20857
Pradip.Muhuri@ahrq.hhs.gov

Charles Z.X. Han
Ryman Healthcare Ltd.
charleszhouxiaohan@gmail.com

John Vickery
North Carolina State University Libraries
(919) 513-0344
John_vickery@ncsu.edu

/* APPENDIX I (SAS Program) */

```
%let path = c:\SESUG_2023;
/* Step 1: Fetch the main MEPS-HC web page's contents using PROC HTTP */
filename source temp;
proc http
    url = "https://meps.ahrq.gov/data_stats/download_data_files.jsp"
    out=source;
run;

/* Step 2:  Parse the MEPS-HC main web page */
/* Get years from dropdown options list */
options generic;
data year_values;
   length year 8;
   infile source dlm='<' expandtabs truncover;
   input @'<option value="'   @'>' year ??;
if not missing (year);
run;

filename source clear;

/* Step 3: Call Proc HTTP for each year_url */
%macro get_year(year);
        filename yearresp "&path\hc_response.txt";
        %local base_url data_year_param search_param year_url;
        /* URL elements */
        %let base_url = https://meps.ahrq.gov/data_stats/download_data_files_results.jsp?;
        %let data_year_param = cboDataYear=&year.;
        %let search_param = %nrstr(&buttonYearandDataType=Search);

        /* Combine the URL elements to construct the year url*/
        %let year_url = &base_url.&data_year_param.&search_param.;

        /* Call PROC HTTP to retrieve the content */
        /* of each year search results */
        proc http
                url = "&year_url."
        out=yearresp;
        run;

        data year_list_&year. (keep=puf_num meps_file data_year);
                length puf_num $15 meps_file $150 data_year $10;
                infile yearresp length = reclen lrecl = 32767 end=eof;

                /* regex to get the PUF num in the table of results */
                retain prx_puf;
        if _n_= 1 then prx_puf = prxparse('/<a href="download_data_files_detail\.jsp\?cboPufNumber=.+\">(.+)<\/a>/');

                /* regex to get the meps file */
        retain prx_meps;
        if _n_=1 then prx_meps = prxparse('/<td height="30" align="left" valign="top" class="bottomRightgrayBorder"><div align="left" class="contentStyle">(.+)<\/font>/');

                /* regex for data year */
                retain prx_data_year;
        if _n_ = 1 then prx_data_year = prxparse('/<td width="1" height="30" align="left" valign="top" class="bottomRightgrayBorder"><div align="left" class="contentStyle">(.+)<\/font><\/div><\/td>/');

                /* Read the HTML line by line */
                do while (not eof);
                        input html_line $varying32767. reclen;
                        if prxmatch(prx_puf, html_line) > 0 then do;
                                call prxposn(prx_puf, 1, start, end);
                        puf_num = substr(html_line, start, end);
                        end;
                  if prxmatch(prx_meps, html_line) > 0 then do;
                        call prxposn(prx_meps, 1, start, end);
                        meps_file = substr(html_line, start, end);
                        meps_file = prxchange('s/<.+>//', -1, meps_file);          /* remove any stray html tags */
                  end;
                  if prxmatch(prx_data_year, html_line) > 0 then do;
                                call prxposn(prx_data_year, 1, start, end);
                        data_year = substr(html_line, start, end);
                        output year_list_&year.;
                        end;
                end;
    run;

    filename yearresp clear;

%mend get_year;

/* Loop through each year of the year_values dataset and call the macro */
data _null_;
    set year_values;
  call execute('%nrstr(%get_year('||strip(year)||'));');
run;

/* Concatenate all year_list_YEAR datasets */
/* Just keep obs with puf_num beginning HC- */
data year_list;
    set year_list_: ;
    where substr(puf_num, 1, 2) = 'HC';
run;

/* De-dup */
proc sort data=year_list nodupkey;
 by _ALL_;
run;

/* Step 4: Get the zip files for each HC- puf_num. */
%macro get_puf(pufnum);
   %local base_url puf_url;
    filename pufresp temp;

    /* URL elements */
    %let base_url =
https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=;

    /* Combine the URL elements to construct the PUF url */
    %let puf_url = &base_url.&pufnum.;

    proc http
            url="&puf_url."
            out= pufresp;
    run;

    data puf_list_%sysfunc(translate(&pufnum, '_', '-')) (keep=puf_num file_format zip_link);
            length puf_num $15 file_format $150 zip_link $200;
            infile pufresp length = reclen lrecl = 32767 end=eof;

     /* regex for the data file name */
            retain prx_data_file;
     if _n_= 1 then prx_data_file = prxparse('/<td width="50%" height="0" class="bottomRightgrayBorder">Data
File.*, (.+)<\/td>/');

            /* regex for the zip file download link */
            retain prx_zip;
     if _n_= 1 then prx_zip = prxparse('/<a href="\.\.\/(data_files[\/pufs]*\/.+\.zip)">ZIP<\/a>/');

            do while (not eof);
                        input html_line $varying32767. reclen;
                        puf_num = "&pufnum.";
                        if prxmatch(prx_data_file, html_line) > 0 then do;
                                call prxposn(prx_data_file, 1, start, end);
                        file_format = substr(html_line, start, end);
                        end;

                        if prxmatch(prx_zip, html_line) > 0 then do;
                                call prxposn(prx_zip, 1, start, end);
                        zip_link = cats("https://meps.ahrq.gov/", substr(html_line, start, end));
                        output puf_list_%sysfunc(translate(&pufnum, '_', '-'));
                        end;
            end;

     run;

    filename pufresp clear;

 %mend get_puf;

 /* Loop through each puf_num of the year_list dataset and call the macro */
 data _null_;
    set year_list;
    call execute('%nrstr(%get_puf('||strip(puf_num)||'));');
 run;

 /* Concatenate all puf_list datasets */
 data puf_list;
    set puf_list_: ;
 run;

 /* De-dup */
 proc sort data=puf_list nodupkey;
    by _ALL_;
 run;

 /* Step 5: Merge year_list and puf_list */
 proc sql;
    create table meps_zip_links as
    select y.puf_num, y.meps_file, y.data_year, p.file_format, p.zip_link
    from year_list as y,
             puf_list as p
    where y.puf_num = p.puf_num
   order by data_year desc, puf_num, file_format;
 quit;

 /* Format the current date as "YYYY-MM-DD" */
 %let current_date = %sysfunc(today());
 %let formatted_date = %sysfunc(putn(&current_date., yymmdd10.));

 /* Step 6: Direct proc report output to excel */
 ods listing close;
 ods excel file = "&path\SAS_Solution_WS_&formatted_date..xlsx"
   options (sheet_name = 'Sheet1'
   flow="header,data" row_heights = '15'
   absolute_column_width='11,11,70,30,55');
 proc report data=meps_zip_links;
  column data_year puf_num meps_file file_format  zip_link;
  define puf_num / display ;
  define meps_file / display;
  define data_year / display;
  define file_format / display;
  define zip_link / display ;
  compute zip_link ;
    call define(_col_,"url",zip_link);
  endcomp;
run;
ods excel close;
ods listing;

```

APPENDIX II (Part of the Excel file from the PROC REPORT Output)

data_year	puf_num	meps_file	file_format	zip_link
2021	HC-227	2021 Jobs File	ASCII format	https://meps.ahrq.gov/data_files/pufs/h227/h227dat.zip
2021	HC-227	2021 Jobs File	SAS V9 format	https://meps.ahrq.gov/data_files/pufs/h227/h227v9.zip
2021	HC-227	2021 Jobs File	SAS transport format	https://meps.ahrq.gov/data_files/pufs/h227/h227ssp.zip
2021	HC-227	2021 Jobs File	Stata format	https://meps.ahrq.gov/data_files/pufs/h227/h227dta.zip
2021	HC-227	2021 Jobs File	XLSX format	https://meps.ahrq.gov/data_files/pufs/h227/h227xlsx.zip
2021	HC-228	2021 Full Year Population Characteristics File	ASCII format	https://meps.ahrq.gov/data_files/pufs/h228/h228dat.zip
2021	HC-228	2021 Full Year Population Characteristics File	SAS V9 format	https://meps.ahrq.gov/data_files/pufs/h228/h228v9.zip
2021	HC-228	2021 Full Year Population Characteristics File	SAS transport format	https://meps.ahrq.gov/data_files/pufs/h228/h228ssp.zip
2021	HC-228	2021 Full Year Population Characteristics File	Stata format	https://meps.ahrq.gov/data_files/pufs/h228/h228dta.zip
2021	HC-228	2021 Full Year Population Characteristics File	XLSX format	https://meps.ahrq.gov/data_files/pufs/h228/h228xlsx.zip

Appendix III (Python program)

```
#!/usr/bin/env python
# coding: utf-8

# **step 1**
# - note the python libraries/modules

# import libraries
import pandas as pd
import requests
from bs4 import BeautifulSoup
from urllib.parse import quote

# **step 2**
# - get the list of numerical year options from the main page
# - save as list of tuples with year and url

# main page response
main_page = requests.get("https://meps.ahrq.gov/data_stats/download_data_files.jsp")
main_soup = BeautifulSoup(main_page.text, "html.parser")

# get list of data years and links for each year
# skip the "All years" (i.e. non-digit options)
base_year_url = "https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.jsp?cboDataYear="
year_url_suffix = "&buttonYearandDataType=Search"

year_options = main_soup.select_one('select[name="cboDataYear"]').select("option")

year_url_list = [
    (x.get("value"), base_year_url + quote(x.get("value")) + year_url_suffix)
    for x in year_options
    if x.get("value").isdigit()
]

# **step 3**
# - use the year_url_list list of tuples to get "HC-" zip file links
# - save these in list of dictionaries

base_hc_url = "https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber="
zip_link_list = []
for year, year_url in year_url_list:
    try:
        response = requests.get(year_url)
        response.raise_for_status()
        try:
            dfs = pd.read_html(response.text)
            hc_df = pd.DataFrame()
            for df in dfs:
                if "File(s), Documentation & Codebooks" in df.columns:
                    hc_df = pd.concat([hc_df, df], ignore_index=True)
                    hc_df = hc_df.dropna(subset="PUF no.")
                    hc_df = hc_df.loc[hc_df["PUF no."].str.contains("HC-")]
                    hc_df = hc_df.drop_duplicates(subset="PUF no.")
                    hc_df["hc_url"] = base_hc_url + hc_df["PUF no."]
            if not hc_df.empty:
                hc_df.columns = [
                    "puf_num",
                    "Files",
                    "Data_Update",
                    "Year",
                    "File_Type",
                    "hc_url",
                ]
                for row in hc_df.itertuples():
                    hc_response = requests.get(row.hc_url)
                    hc_response.raise_for_status()
                    hc_soup = BeautifulSoup(hc_response.text, features="lxml")
                    try:
                        meps_file = hc_soup.find(class_="OrangeBox").text
                        for td in hc_soup.find_all("td"):
                            zip_link_dict = {}
                            if td.text.startswith("Data File"):
                                zip_link_dict["data_year"] = row.Year
                                zip_link_dict["puf_num"] = row.puf_num
                                zip_link_dict["meps_file"] = meps_file
                                zip_link_dict["file_format"] = td.text
                                zip_link_dict[
                                    "zip_link"
                                ] = "https://meps.ahrq.gov" + td.find_next("a").get(
                                    "href"
                                ).strip(
                                    ".."
                                )
                                zip_link_list.append(zip_link_dict)
                    except:
                        # catch your exceptions here
                        pass
        except requests.exceptions.HTTPError as err:
            print(err)
    except requests.exceptions.HTTPError as httperr:
        print(httperr)

# **step 4**
# - save the list of dictionaries to pandas dataframe
# - deduplicate
# - clean up meps_file column to remove "PUF no." prefix
# - clean up file_format column to remove "Data File" prefix
# - sort the dataframe in certain order
# - get n rows of the dataframe

meps_df = pd.DataFrame(zip_link_list)
meps_df.drop_duplicates(inplace=True)
meps_df["meps_file"] = meps_df["meps_file"].str.split(": ", n=1).str[-1]
meps_df["file_format"] = meps_df["file_format"].str.split(", ", n=1).str[-1]
meps_df = meps_df.sort_values(by=['data_year','puf_num', 'file_format'], ascending=[False, True, True])
print(meps_df.head(5))

# **step 4 (continued)**
# - obtain the current date (timestamp)
# - save the dataframe as an excel file

today = pd.Timestamp("now").strftime("%Y-%m-%d")
meps_df.to_excel(f"C:\SESUG_2023\Python_Solution_WS_{today}.xlsx", index=False)

```