

One Click to Create a List of 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 case study demonstrates the use of SAS® and Python to automate data extraction and processing from the Medical Expenditure Panel Survey (MEPS) webpages. Both programs streamline the tasks of fetching webpages, parsing the HTML content, and generating identical outputs: a comprehensive list of 1,299 data file download URLs and associated information for 431 MEPS-Household Component (HC) public-use ZIP files in various formats (e.g., ASCII, Excel, SAS transport, SAS V9, and Stata), covering survey years from 1996 to 2022. This list is saved in an Excel spreadsheet, allowing users to download files of interest directly to their devices with a single click on each corresponding URL, offering a faster alternative to manually navigating multiple webpages. The dynamic construction of these URLs and the reuse of the SAS/Python program to keep the list of data file download URLs updated streamline the process, making MEPS data file downloads from the web more efficient.

INTRODUCTION

Over the years, the Agency for Healthcare Research and Quality (AHRQ), on the official website (<https://meps.ahrq.gov>) for the Medical Expenditure Panel Survey (MEPS), has made available various MEPS-Household Component (HC) public-use ZIP files¹. The MEPS website's main webpage, https://meps.ahrq.gov/data_stats/download_data_files.jsp, presents an exhaustive list of these files by survey year and file type².

From the MEPS main webpage, users can sequentially navigate to two sets of webpages: first, those associated with specific survey years, and second, those related to individual public-use files (PUFs) for a given survey year. In the first set, the MEPS year-specific webpage³ provides a comprehensive list of all available PUFs for that year. A unique PUF number identifies each file with an embedded URL, description, data update, and type. In the second set, the MEPS PUF-specific webpage⁴ includes several sections, including documentation and its file type, data file and its type, and survey questionnaires, each with its own embedded link. The data file type section of the PUF webpage provides a list of format-specific data files (e.g., ASCII, Excel, SAS transport, SAS V9, and Stata), and each file has its data download URL⁵. These URLs allow users to quickly download data format-specific PUFs (e.g., a PUF in

¹ The Medical Expenditure Panel Survey (MEPS), conducted annually since 1996 by the AHRQ, consists of two primary components: the Household Component (HC) and the Insurance Component (IC). The MEPS-HC collects nationally representative data from individuals in the U.S. civilian noninstitutionalized population through questionnaires covering demographic characteristics, health conditions, medical service usage, costs, health insurance, access to and satisfaction with care, income, and employment. MEPS-HC public-use files (PUFs) in ZIP format are available online. The MEPS-IC gathers data on employees' health insurance coverage from U.S. private and public sector employers. These data files are not available online and are outside the scope of this case study. We have used the terms 'MEPS-HC public-use ZIP files', 'public-use files (PUFs)', 'ZIP files', and 'MEPS data files' interchangeably throughout the paper to refer to the same data sets.

² The MEPS-HC public-use ZIP files include person-level files like longitudinal and full-year consolidated files. Additionally, they cover various event-level files related to medical conditions, prescribed medicines, and different types of healthcare visits, all organized by survey year from 1996 to 2022.

³ As an example, the webpage for the 2022 survey year on the MEPS website lists all PUFs for that year at https://meps.ahrq.gov/data_stats/download_data_files_results.jsp?cboDataYear=2022&buttonYearandDataType=Search.

⁴ For example, the webpage for the PUF named MEPS HC-226: MEPS Panel 23 Three-Year Longitudinal Data File contains links to the documentation, codebook, programming statements, and format-specific data files for that PUF at https://meps.ahrq.gov/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-226.

⁵ The URL for downloading a MEPS-HC PUF is visible when hovering over the clickable text for a specific format, such as 'ZIP for Data File, SAS transport format.' For example, when hovering over this text, the URL displayed at the

SAS V9 format); however, due to their design, they can't be directly copied and pasted into an external file for later use. This limitation may require alternative methods like automated scripts to effectively capture and organize the PUF download URLs.

Since there is no consolidated list of MEPS data file download URLs available online, finding and downloading multiple files can be tedious and time-consuming. The process often involves manually navigating various webpages described above, which becomes increasingly inefficient as the number of desired files to download grows. This procedure underscores the importance of implementing automated solutions to efficiently manage the increasing volume of online MEPS data files. These solutions would streamline the process and ensure the data file download URLs can be systematically captured, organized, and utilized as needed. The objective of scraping data from the MEPS webpages using SAS® and Python is to dynamically construct an up-to-date list of those URLs for various MEPS-HC public-use ZIP files, along with several identifiers, and to save the list in individual Excel spreadsheets⁶.

SAS® PROGRAM

We present a SAS program, detailed in APPENDIX I, as our first web scraping solution to achieve the stated objective. The program uses various SAS features, including macro programming for code reusability and automation, PROC HTTP for scraping webpages, regular expressions for parsing HTML, PROC SQL for data joins, PROC REPORT for defining and displaying the final content, and ODS EXCEL for creating the ultimate output file. We describe the program in eight steps below. We used SAS® 9.04.01M6 X64_10PRO in the Windowing environment.

Step 1: Setup File Path and Fetch the MEPS Main Webpage

Line 1: The code creates a macro variable PATH and assigns it the value c:\SESUG_2024.

Line 3: The FILENAME statement associates the TEMP device with the file reference (i.e., SAS fileref) SOURCE, creating a temporary location that remains as long as the filename is assigned.

Lines 4-7: The PROC HTTP procedure sends an HTTP GET request to the MEPS main webpage https://meps.ahrq.gov/data_stats/download_data_files.jsp specified in the URL= option and writes the response body to the fileref SOURCE specified in the OUT= option. This response body is the input for the DATA step discussed below.

Step 2: Parse HTML Content to Extract Year

Line 12: The DATA statement creates an output 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 copied from the HTTP response body:

```
<td width="523" height="0"><span class="contentStyle">
    <select id="datayear" size=1 name="cboDataYear">
        <option value="All">All available years</option>
        <option value="2022">2022</option>
        <option value="2021">2021</option>
        ... Some data records are skipped ...
        <option value="1996">1996</option>
```

bottom left-hand corner of the page is: https://meps.ahrq.gov/data_files/pufs/h226/h226ssp.zip.

⁶ The results reported are based on web scraping using SAS and Python done on August 5, 2024, and may vary in future updates. Please see footnote 15 for details.

<option value="Projected Years">Projected Years</option>

Line 15: The `@'character-string'` syntax⁷ moves the input pointer to the correct location in the HTML data record to read the content between opening `<option>` and closing `</option>` tags. The content is simply the value (e.g., 1996, ..., 2021, and 2022) of the HTML `<option>` value attribute assigned to the SAS 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.

Lines 16-17: The code instructs SAS to keep only the observations with non-missing values for the variable YEAR and then execute the DATA step, creating the data set YEAR_VALUES.

The SAS Log:

```
NOTE: 3346 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 27 observations and 1 variables.
```

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

Step 3: Retrieve Data from Each MEPS Year Webpage

For this task, we use the `%GET_YEAR` macro, which loops over YEAR values in the YEAR_VALUES data set and constructs year-specific URLs.

The macro-generated SAS code is intended to perform the following tasks for years from 1996 to 2022:

- Fetch the content of the year-specific MEPS webpage.
- Parse the retrieved HTML content.
- Use Perl regular expressions (regex) to extract
 - PUF number (e.g., HC-240),
 - file name (e.g., 2022 Food Security File) and
 - data year (e.g., 2022).
- Save the created variables in an output data set.

Below are some details about the `%GET_YEAR` macro definition:

Lines 22: As a first statement in the macro definition, the `%MACRO` statement includes its macro name `GET_YEAR`, adding a positional macro parameter, YEAR, in parentheses.

Line 23: The FILENAME statement associates the TEMP device with the file reference (i.e., SAS fileref) YEARRESP, creating a temporary location that remains as long as the filename is assigned.

Lines 24-31: The `%LET` statements create local macro variables (`BASE_URL`, `DATA_YEAR_PARAM`, and `SEARCH_PARAM`) for three URL components. The subsequent `%LET` statement creates a new macro variable (`YEAR_URL`) by concatenating those three macro variable references, effectively placing them next to each other. When the `%GET_YEAR` macro is executed with a value to its parameter YEAR, the macro variable reference `&YEAR_URL` resolves to the corresponding valid YEAR URL. For example,

⁷ For more details, please refer to the `@'character-string'` section available at https://go.documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lestmtsref/n0oagql83drile0n141pdacojq97s.htm#p10kxp5ghaqig7n1qcmwle2swfox.

the macro call %GET_YEAR(1996) causes the parameter &YEAR_URL to resolve to https://meps.ahrq.gov/data_stats/download_data_files_results.jsp?cboDataYear=1996&buttonYearandDataType=Search.

Lines 35-38: The PROC HTTP procedure sends an HTTP GET request to the URL specified by the macro variable reference &YEAR_URL, using the URL= option. It then writes the response body to the fileref YEARRESP specified in the OUT= option.

Lines 40-42: The DATA statement specifies the data set name using the macro variable YEAR_LIST_&YEAR, where &YEAR is dynamically replaced with its value when the %GET_YEAR macro is called with the YEAR parameter. The LENGTH statement defines the length of the character variables specified. The INFILE statement specifies the fileref YEARRESP to link to the raw data file with the LENGTH=RECL option to set the record length of each line to a variable named RECL, the LRECL=32767 option to set the logical record length to a maximum of 32,767 bytes, and the END=EOF option to create a boolean variable EOF that will be set to 1 when the end of the file is reached, otherwise 0.

Lines 45-54: The RETAIN statement retains the value of the compiled version of the regex pattern across iterations. The PRXPARSE function in the 'IF _N_ = 1 then ...' statement compiles each of the three regex patterns specified between the first and last forward slashes with single quotes in parentheses. Below is a description of the behavior of the metacharacters used in three regex patterns:

- \. matches the literal period.
- \? matches the literal question mark.
- \/ matches the literal forward slash.
- .+ matches any single character one or more times.
- (.+) creates a capture buffer.

Each of the three compiled regex patterns created by the PRXPARSE function returns a value (a non-zero positive integer that serves as a pattern identifier). The variables that hold the three compiled regex patterns are as follows:

- PRX_PUF for the PUF number,
- PRX_MEPS for the file name, and
- PRX_DATA_YEAR for the data year.

Lines 57-73: The DO WHILE (NOT EOF) begins a loop that executes as long as the end of file condition is not met. The INPUT statement includes a character variable named HTML_LINE to read values of the raw data field with an informat (\$VARYING32767.) and another variable named RECL that stores the actual length of the line being read. A conditional statement (i.e., IF-THEN DO) using the PRXMATCH function checks whether each of the three regular expression patterns matches individually against the HTML_LINE values. If a match is found, the CALL PRXPOSN routine uses the results of PRXMATCH to return the starting and ending positions (values) within the three captured content of the HTML_LINE variable separately. The SUBSTR function uses these positions to extract the captured content to construct three variables: PUF_NUM, MEPS_FILE, and DATA_YEAR. The PRXCHANGE function is used to remove any remaining HTML tags in the MEPS_FILE value. The DATA step writes the observation to the output data set only when it successfully creates the corresponding variable for that observation.

Line 76: The FILENAME statement with the CLEAR option disassociates the fileref YEARRESP.

Line 78: The %MEND statement marks the end of the macro definition.

Lines 81-84: In the DATA_NULL_step, CALL EXECUTE is used to generate and execute the %GET_YEAR macro call with its parameter YEAR (whose values are obtained from the YEAR variable in

the YEAR_VALUES data set). The %NRSTR macro quoting function prevents immediate macro resolution when the macro is compiled, delaying the execution of the macro until after the DATA step boundary [2].

The partial SAS log⁸ from a %GET_YEAR macro call with a parameter value of 2022:

NOTE: CALL EXECUTE generated line.

```
1 + %get_year(2022);
```

NOTE: PROCEDURE HTTP used (Total process time):

```
real time          0.02 seconds
```

```
cpu time           0.00 seconds
```

NOTE: 200 OK

NOTE: The infile YEARRESP is:

```
(system-specific pathname),
```

```
(system-specific file attributes)
```

NOTE: DATA STEP stopped due to looping.

NOTE: 1122 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_LIST_2022 has 12 observations and 3 variables.

Step 4: Combine and Filter Year-Specific Webpage Data

Lines 88-91: The DATA step concatenates all year-specific webpage data sets (e.g., YEAR_LIST_1996, YEAR_LIST_1997, ... YEAR_LIST_2022) using the name prefix (YEAR_LIST_) followed by a colon(:) to select those ending in numeric suffixes (i.e., 1996, ... 2022) in the SET statement. SAS selects observations for the output data set where the SUBSTR function in the WHERE statement returns the value 'HC' (a substring of the PUF_NUM variable value). This output SAS data set (YEAR_LIST⁹) includes observations with a valid MEPS-HC PUF number.

Lines 94-96: The NODUPKEY option in the PROC SORT statement prevents an observation from duplicating in the output data set if its BY value matches that of the previous observation written to the data set YEAR_LIST. In other words, if two observations have the same BY value, only one is retained in the output data set¹⁰, with all variable values considered.

The partial SAS log (with some notes skipped):

NOTE: The data set WORK.YEAR_LIST has 539 observations and 3 variables.

NOTE: 85 observations with duplicate key values were deleted.

NOTE: The data set WORK.YEAR_LIST has 454 observations and 3 variables.

The PROC PRINT code¹¹ (followed by the corresponding output):

```
proc print data=work.year_list noobs;  
  where data_year='2022';  
run;
```

puf_num	meps_file	data_year
---------	-----------	-----------

⁸ After copying part of the original SAS log here, some blank lines between notes were deleted.

⁹ There are 539 observations in the WORK.YEAR_LIST data set. Some observations are duplicates.

¹⁰ Deduplication of identical observations for repeated values of the PUF_NUM, MEPS_FILE, and DATA_YEAR variables resulted in the exclusion of 85 observations. Here is an example of an excluded observation with three variables and their values in parentheses: PUF_NUM (HC-235), MEPS_FILE (MEPS Panel 24 Three-Year Longitudinal Data File, and DATA_YEAR (2019-2020). The log provided indicates that the total number of observations remaining after deduplication is 454.

¹¹ The PROC PRINT code used for this output is not included in the the SAS program in APPENDIX I.

HC-240	2022 Food Security File	2022
HC-239H	2022 Home Health File	2022
HC-239G	2022 Office-Based Medical Provider Visits File	2022
HC-239F	2022 Outpatient Visits File	2022
HC-239E	2022 Emergency Room Visits File	2022
HC-239D	2022 Hospital Inpatient Stays File	2022
HC-239C	2022 Other Medical Expenses File	2022
HC-239B	2022 Dental Visits File	2022
HC-239A	2022 Prescribed Medicines File	2022
HC-238	2022 Full Year Population Characteristics	2022
HC-237	2022 Jobs File	2022

Output 1: PROC PRINT listing from a subset of WORK.YEAR_LIST

The PUF_NUM values in the YEAR_LIST dataset are passed to the macro parameter for another macro that we will discuss next, followed by PROC SQL joins. The YEAR_LIST dataset is one of the input data sets in these joins.

Step 5: Retrieve Data from Each MEPS PUF Webpage

For this task, we use the %GET_PUF macro, which generates SAS code that fetches the contents of the PUF-specific MEPS webpage, parses the retrieved HTML content, and creates variables with additional information about MEPS-HC PUFs across different data years. This information includes:

- PUF file format (e.g., SAS V9) and
- ZIP file link (e.g., https://meps.ahrq.gov/data_files/pufs/h237/h237v9.zip).

Below are some details about the %GET_PUF macro definition:

Lines 99: As a first statement in the macro definition, the %MACRO statement includes its macro name GET_PUF, adding a positional macro parameter, PUFNUM, in parentheses.

Line 101: The FILENAME statement associates the TEMP device to the file reference (i.e., SAS fileref) PUFRESP, creating a temporary location that remains as long as the filename is assigned.

Lines 104-107: The %LET statement creates a local macro variable (BASE_URL). The subsequent %LET statement creates a new PUF-specific macro variable (PUF_URL) by concatenating the two macro variable references &BASE_URL and &PUFNUM, effectively placing them next to each other. When the %GET_PUF macro is executed with a value to its parameter PUFNUM, the macro variable reference &PUF_URL resolves to the corresponding valid PUF URL. For example, the macro call %GET_PUF(HC-012) causes the parameter &PUF_URL to resolve to https://meps.ahrq.gov/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-012.

Lines 109-112: The PROC HTTP procedure sends an HTTP GET request to the URL specified by the macro variable reference &PUFNUM, using the URL= option. It then writes the response body to the fileref PUFRESP specified in the OUT= option.

Lines 114-116: The DATA statement specifies the data set name beginning with PUF_LIST_ followed by the value of &PUFNUM with any hyphen converted to an underscore using the %SYSFUNC to allow us to use the TRANSLATE function in a macro context. The KEEP= data set option specifies only the variables PUF_NUM, FILE_FORMAT, and ZIP_LINK to be kept in the output data set. The LENGTH statement defines the length of the character variables specified. The INFILE statement specifies the fileref PUFRESP to link to the raw data file with similar options used in the YEARRESP in the %GET_YEAR macro definition above. Therefore, we will not reiterate their explanations in the context of the %GET_PUF macro definition.

Lines 118-124: As before, the PRXPARSE function is used to create variables that hold two compiled regex patterns:

- PRX_DATA_FILE for data format
- PRX_ZIP for ZIP file download link

The regex pattern used in the PRXPARSE function's argument includes the metacharacters .+ and (.+), whose behaviors were described earlier. The extra metacharacters .* indicate zero or more occurrences of any character in the PRX_DATA_FILE regex. The character class groupings [/pufs] match either a forward slash (/) or the word "pufs" in the PRX_ZIP regex.

Lines 126-141: The DO WHILE (NOT EOF) begins a loop that executes as long as the end of file condition is not met. The INPUT statement includes a character variable named HTML_LINE to read values of the raw data field with an informat (\$VARYING32767.) and another variable named RECLLEN that stores the actual length of the line being read. In a string context, the SAS assignment statement creates a character variable PUF_NUM by referencing the PUFNUM macro parameter on the right-hand side of the equation. The code block utilizes the PRXMATCH function, the CALL PRXPOSN routine, and the SUBSTR function to generate variables PUF_NUM, FILE_FORMAT, and ZIP_LINK (example values displayed in Output 2). When describing the %GET_YEAR macro definition above, we explained these functions and the routine. Therefore, we will not reiterate their explanations in the context of the %GET_PUF macro definition. The DATA step only writes the observation to the output data set if it finds the variable created for that observation.

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

Line 145: The %MEND statement marks the end of the macro definition.

Lines 148-151: In the DATA _NULL_ step, CALL EXECUTE is used to generate and execute the %GET_PUF macro call with its parameter PUFNUM (values obtained from the PUF_NUM variable in the YEAR_LIST data set) without hard-coding each public use file number. The %NRSTR macro quoting function prevents immediate macro resolution when the macro is compiled, delaying the execution of the macro until after the DATA step boundary [2]. As a result, the %GET_PUF macro generates SAS code to create 1,299 data file download URLs associated with 431 MEPS-HC public-use ZIP files in various formats (e.g., ASCII, Excel, SAS transport, SAS V9, and Stata), covering survey years from 1996 through 2022 by retrieving the HTML content from the corresponding webpages.

The partial SAS log¹² from a %GET_PUF macro call with its parameter value of HC-228:

```
427 + %get_puf(HC-228);

NOTE: PROCEDURE HTTP used (Total process time):
      real time           0.02 seconds
      cpu time            0.00 seconds
NOTE: 200 OK
NOTE: The infile PUFRESP is:
      (system-specific pathname),
      (system-specific file attributes)
NOTE: DATA STEP stopped due to looping.
NOTE: 1171 records were read from the infile (system-specific pathname).
      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 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.08 seconds
      cpu time            0.04 seconds

NOTE: Fileref PUFRESP has been deassigned.
```

¹² After copying part of the original SAS log here, some blank lines between notes are deleted.

Step 6: Combine and Filter PUF Webpage Data

Lines 154-156: The DATA step concatenates the SAS data sets using the name prefix (PUF_LIST_) followed by a colon(:) to select those ending specific patterns as suffixes (e.g., HC_228). The resulting output data set is PUF_LIST with 1,299 observations (ZIP_LINK values from 431¹³ MEPS-HC PUFs).

Lines 159-161: The NODUPKEY option in the PROC SORT statement prevents an observation from duplicating in the output data set if its BY value matches that of the previous observation written to the data set PUF_LIST. In other words, if two observations have the same BY value, only one is retained in the output data set, with all variable values considered.

The partial SAS log (followed by some other partial SAS output):

NOTE: 0 observations with duplicate key values were deleted.

NOTE: The data set WORK.PUF_LIST has 1299 observations and 3 variables.

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

Output 2: Partial PROC PRINT listing from WORK.PUF_LIST_HC_228¹⁴

Step 7: Join Year- and PUF-Specific Webpage Data

In this step, PROC SQL combines two data tables using joins with a WHERE clause to create an output data table ordered by specific columns.

Lines 164-171: The SQL procedure performs a one-to-many match between the tables YEAR_LIST and PUF_LIST using table aliases in the FROM clause. This match is based on the column PUF_NUM, which is common to both tables. The created table MEPS_ZIP_LINKS is ordered by the column DATA_YEAR in descending order and by PUF_NUM and FILE_FORMAT in ascending order.

- The table YEAR_LIST has a unique value for the column PUF_NUM.
- The table PUF_LIST includes multiple rows with the same value for the column PUF_NUM.
- The output table MEPS_ZIP_LINKS contains the columns for DATA_YEAR, PUF_NUM, MEPS_FILE, FILE_FORMAT, and ZIP_LINK (with a unique value of ZIP file download URL).

The SAS log:

NOTE: Table WORK.MEPS_ZIP_LINKS created, with 1299 rows and 5 columns.

Step 8: Export Final Results to Excel

Line 174: The SAS statement using the %SYSFUNC macro function assigns the current date, formatted

¹³ This count excludes 23 of the 454 observations in the YEAR_LIST data set because those excluded observations have missing values on the ZIP_LINK variable. The corresponding 23 PUFs were ad-hoc files and were later updated with new PUF numbers and data download URLs on the webpage. Thus, old data download URLs no longer exist on the webpage, so ZIP_LINK values are missing. Here are examples of old and “replaced” URLs: https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-002 (HC-012 replaced HC-002) and https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-012.

¹⁴ The PROC PRINT code, used to generate the output, is not part of the SAS program in APPENDIX I.

as desired, to the macro variable CURRENT_DATE.

Line 175: This line of code formats the value of the CURRENT_DATE macro variable according to the specified date format and assigns it to a new macro variable named FORMATTED_DATE.

Lines 179-194: The ODS EXCEL statement directs the PROC REPORT output to a file specified in the FILE=option, dynamically generating an Excel file name that includes the current date. The REPORT procedure uses the contents of the SAS data set MEPS_ZIP_LINKS to display the values of the specified variables. The CALL DEFINE associates the URL with the ZIP_LINK variable. An Excel file is created after executing the ODS EXCEL CLOSE; statement. This spreadsheet includes 1,299 unique URLs¹⁵ for downloading 431 MEPS-HC public-use ZIP files in various data formats (i.e., ASCII, Excel, SAS transport, SAS V9, and Stata), covering survey years from 1996 to 2022. Users can easily download the files of interest to their devices with a single click on each corresponding URL in the Excel spreadsheet.

data_year	puf_num	meps_file	file_format	zip_link
2022	HC-237	2022 Jobs File	ASCII format	https://meps.ahrq.gov/data_files/pufs/h237/h237dat.zip
2022	HC-237	2022 Jobs File	SAS V9 format	https://meps.ahrq.gov/data_files/pufs/h237/h237v9.zip
2022	HC-237	2022 Jobs File	SAS transport format	https://meps.ahrq.gov/data_files/pufs/h237/h237ssp.zip
2022	HC-237	2022 Jobs File	Stata format	https://meps.ahrq.gov/data_files/pufs/h237/h237dta.zip
2022	HC-237	2022 Jobs File	XLSX format	https://meps.ahrq.gov/data_files/pufs/h237/h237xlsx.zip
2022	HC-238	2022 Full Year Population Characteristics	ASCII format	https://meps.ahrq.gov/data_files/pufs/h238/h238dat.zip
2022	HC-238	2022 Full Year Population Characteristics	SAS V9 format	https://meps.ahrq.gov/data_files/pufs/h238/h238v9.zip
2022	HC-238	2022 Full Year Population Characteristics	SAS transport format	https://meps.ahrq.gov/data_files/pufs/h238/h238ssp.zip
2022	HC-238	2022 Full Year Population Characteristics	Stata format	https://meps.ahrq.gov/data_files/pufs/h238/h238dta.zip
2022	HC-238	2022 Full Year Population Characteristics	XLSX format	https://meps.ahrq.gov/data_files/pufs/h238/h238xlsx.zip

Output 3: Partial PROC REPORT listing

PYTHON PROGRAM

We present a Python program, detailed in APPENDIX II, as our second web scraping solution for the objective outlined earlier. The script includes error handling to manage potential HTTP errors or parsing problems so that the script continues running in the event of failing to retrieve some data. We describe the program in five steps below. We used Python Version 3.9.12 in the Spyder IDE Version 5.1.5.

Step 1: Import libraries

Lines 8-11: The program imports necessary libraries, including Requests and BeautifulSoup.¹⁶

Step 2: Fetch MEPS Main Webpage and Create Year URL List

In this step, the Python script is intended to do the following:

- Send a request to the MEPS main page URL.
- Parse HTML content.
- Retrieve the list of numerical year options.

¹⁵ The results reported are based on web scraping using SAS and Python done on August 5, 2024, and may vary in future updates. An increase in the number of data file download URLs is expected by the end of 2024 as more 2022 MEPS data files are posted online. Typically, there is a lag of about two years between MEPS-HC data collection and the release of the public-use files.

¹⁶ BeautifulSoup, Scrapy, and Selenium are essential Python tools for web scraping. BeautifulSoup (from the Python bs4 library) extracts data from single webpages. Scrapy crawls multiple webpages, downloading and parsing data. Selenium can automate webpage navigation and dynamic content scraping.

- Store the data in a list of tuples, with each tuple containing the year and the corresponding URL.

Lines 18-19: This code uses the `requests` library to send a GET request to a webpage with the URL "https://meps.ahrq.gov/data_stats/download_data_files.jsp". The response from the webpage 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. Below is the additional code (not shown in the Python program in APPENDIX II) to print the Python list `year\_options` followed by the output:

```
print(year_options)
```

```
[<option value="All">All available years</option>, <option value="2022">2022</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">2012</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">2001</option>, <option value="2000">2000</option>, <option value="1999">1999</option>, <option value="1998">1998</option>, <option value="1997">1997</option>, <option value="1996">1996</option>, <option value="Projected Years">Projected Years</option>]
```

Output 4: Python list `year\_options`

Below is the one-liner code (not shown in the APPENDIX II Python program) to print the first two elements of the Python list of tuples `year\_url\_list`, including the elements as Python tuples and their corresponding output.

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

```
('2022', 'https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.jsp?cboDataYear=2022&buttonYearandDataType=Search')
('2021', 'https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.jsp?cboDataYear=2021&buttonYearandDataType=Search')
```

Output 5: Two elements of the Python list of tuples `year\_url\_list`

Step 3: Fetch Year URLs and Extract Information

In this step, the Python script is designed to accomplish the following tasks:

- Loop over year URLs.
- Fetch each year's URL.
- Parse HTML table and create public-use file (PUF) URLs
- Loop over PUF URLs
- Fetch each PUF URL page.

- Extract information (i.e., Data Year, PUF Number, MEPS File, File Format, and ZIP Link)
- Create a dictionary in Python (with key-value pairs with extracted information) for each PUF.
- Append to a Python list of dictionaries.

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. 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-HC 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.iteruples()``. For each row, it sends a GET request to the MEPS-HC PUF URL (``row.hc_url``), raises an exception if the request fails and parses the HTML response using ``BeautifulSoup``. In Python programming, it is a common practice to use exception handling with the ``try-except`` block to catch exceptions that may arise in situations with the possibility of errors, for example, the data year being out of the range in MEPS webpages.

Lines 68-83: If successful, the code extracts the name of the MEPS-HC PUF 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``. Below is the one-liner code (not shown in the Python program in APPENDIX II) to print the first two elements of the Python list of dictionaries ``zip_link_list``, with its elements as the Python dictionary followed by the output:

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

{'data_year': '2022', 'puf_num': 'HC-238', 'meps_file': 'MEPS HC-238: 2022 Full Year Population Characteristics', 'file_format': 'Data File, ASCII format', 'zip_link': 'https://meps.ahrq.gov/data_files/pufs/h238/h238dat.zip'}
{'data_year': '2022', 'puf_num': 'HC-238', 'meps_file': 'MEPS HC-238: 2022 Full Year Population Characteristics', 'file_format': 'Data File, SAS transport format', 'zip_link': 'https://meps.ahrq.gov/data_files/pufs/h238/h238ssp.zip'}
```

Output 6: Two elements of the Python list of dictionaries ``zip_link_list``

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: Process and Save Data

In this step, the Python script aims to perform the following tasks:

- Convert the ZIP file-download link list to a DataFrame.
- Remove duplicate rows from the DataFrame.
- Clean up the columns and sort the DataFrame.
- Print a few rows of the DataFrame.

Line 100: The code first creates the DataFrame `meps_df` using the `pd.DataFrame()` constructor and passing `zip_link_list` as an argument containing 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.

Step 5: Export Data to Excel

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_2024\Python_Solution_WS_YYYY-MM-DD.xlsx".

COMPARISON OF SAS AND PYTHON OUTPUT

The Python script below reads Excel files¹⁷ generated by SAS and Python into pandas DataFrames using the `read_excel()` method. It then compares the two DataFrames using the `compare()` method available in

¹⁷ We manually removed the !headers and footers from the SAS-generated Excel file (SAS_Solution_WS_2024-08-05.xlsx) using the Excel Document Inspector before comparing it with the Python-generated Excel file using the Python script mentioned above (not the Python program in APPENDIX II). After this manual removal, no warnings were issued in the IPython Console during the comparison. However, when we ran the same Python script without addressing the !headers and footers in the SAS-generated file, the script produced the desired output but issued the following warning in the IPython Console: `header_footer.py:48: UserWarning: Cannot parse header or footer so it will be ignored.`

pandas version 1.4. The corresponding IPython console output is shown afterward.

```
import pandas as pd
df1 = pd.read_excel(r"C:\SESUG_2024\SAS_Solution_WS_2024-08-05.xlsx")
df2 = pd.read_excel(r"C:\SESUG_2024\Python_Solution_WS_2024-08-05.xlsx")
try:
    df1.compare(df2).rename({'self': 'left', 'other': 'right'}, axis=1, level=1)
    print ('All rows from the SAS- and Python-created Excel files match')
except:
    print("We caught an error")
```

IPython 8.2.0 -- An enhanced Interactive Python.

```
runfile('C:/Python/Web_Scraping/compare_2024.py', wdir='C:/Python/Web_Scraping')
All rows from the SAS- and Python-created Excel files match
```

Output 7: Results from the IPython console

The comparison confirms that the lists of URLs for MEPS-HC public-use ZIP files generated by both SAS and Python are identical in the Excel files. This consistency ensures that either application can be reliably used to produce accurate and up-to-date lists of data download URLs.

CONCLUSION

For those experienced with Base SAS and macro programming, the SAS program presented here provides an efficient solution for scraping data from MEPS webpages. Modular macros allow for the generation of PROC HTTP and DATA step code across multiple survey years and public-use file numbers, eliminating the need to write similar code blocks repeatedly. In the context of MEPS webpages, these macros significantly simplify the web scraping process. However, Python enthusiasts might favor a Python-based approach due to the language's dynamic features, essential libraries, and overall flexibility. Python's ease of use and minimal coding requirements often make it a more efficient alternative to the corresponding SAS program.

As this case study suggests, SAS and Python generate identical outputs: a comprehensive list of MEPS data file download URLs and associated information in an Excel spreadsheet¹⁸. The list allows for quick and easy downloading of MEPS-HC public-use ZIP files of interest directly from the spreadsheet with a single click on each corresponding URL, offering a faster alternative to manually navigating multiple webpages. Additionally, either program can be reused to maintain an up-to-date list of URLs and related information as new public-use files are added to the MEPS website from the ongoing panel survey, reflecting periodic and annual updates.

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.
2. SAS Institute Inc. 2023. SAS® Help Center. Macro Language: CALL EXECUTE ROUTINE. <https://documentation.sas.com/doc/en/vdmmlcdc/8.1/mcrolref/n1q1527d51eivsn1ob5hznz0yd1hx.htm>.

ACKNOWLEDGMENTS

¹⁸ The results reported are based on web scraping using SAS and Python done on August 5, 2024, and may vary in future updates. Please see footnote 15 for details.

The views expressed in this paper are the authors' and do not imply any official endorsement by the Department of Health and Human Services or the Agency for Healthcare Research and Quality (AHRQ). However, the authors thank Joel Cohen, Frederick Rohde, Fatma Yildirim at AHRQ, and Kurt Bremser in SAS Support Communities for reviewing earlier manuscript versions and providing valuable comments. The authors would like to acknowledge the assistance of OpenAI's ChatGPT in refining the language and structure of parts of this document. Finally, the authors are responsible for any errors or omissions.

Contribution Statements: Pradip K. Muhuri conceived the paper's idea, drafted it, and wrote the initial version of the SAS and Python programs. Charles Z.X. Han revised the Python program and contributed to the code description. Later, John Vickery optimized the SAS and Python programs to their current form. Pradip K. Muhuri then worked on further manuscript revisions with Charles Z.X.'s and John Vickery's contributions.

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

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.

APPENDIX I (SAS Program¹⁹)

```
1 %let path = c:\SESUG_2024;
2 /* Step 1: Fetch the main MEPS-HC Webpage's contents using PROC HTTP */
3 filename source temp;
4 proc http
5     url = "https://meps.ahrq.gov/data_stats/download_data_files.jsp"
6     out=source;
7 run;
8
9 /* Step 2: Parse the MEPS-HC main Webpage */
10 /* Get years from dropdown options list */
11 options generic;
12 data year_values;
13     length year 8;
14     infile source dlm='<' expandtabs truncover;
15     input @'<option value=' @'>' year ??;
16 if not missing (year);
17 run;
18
19 filename source clear;
20
21 /* Step 3: Call Proc HTTP for each year_url */
22 %macro get_year(year);
23     filename yearresp "&path\hc_response.txt";
24     %local base_url data_year_param search_param year_url;
25     /* URL elements */
26     %let base_url = https://meps.ahrq.gov/data_stats/download_data_files_results.jsp?;
27     %let data_year_param = cboDataYear=&year.;
28     %let search_param = %nrstr(&buttonYearandDataType=Search);
29
30     /* Combine the URL elements to construct the year url*/
31     %let year_url = &base_url.&data_year_param.&search_param.;
32
33     /* Call PROC HTTP to retrieve the content */
34     /* of each year search results */
35     proc http
36         url = "&year_url."
37         out=yearresp;
38     run;
39
40     data year_list_&year (keep=puf_num meps_file data_year);
41         length puf_num $15 meps_file $150 data_year $10;
42         infile yearresp length = recln lrecl = 32767 end=eof;
43
44         /* regex to get the PUF num in the table of results */
45         retain prx_puf;
46         if _n_ = 1 then prx_puf = prxparse('/<a
href="download_data_files_detail.jsp?cboPufNumber=.+\ ">(.+)</a>');
47
48         /* regex to get the meps file */
49         retain prx_meps;
50         if _n_ = 1 then prx_meps = prxparse('/<td height="30" align="left" valign="top"
class="bottomRightgrayBorder"><div align="left" class="contentStyle">(.+)</div>');
51
52         /* regex for data year */
53         retain prx_data_year;
54         if _n_ = 1 then prx_data_year = prxparse('/<td width="1" height="30" align="left" valign="top"
class="bottomRightgrayBorder"><div align="left" class="contentStyle">(.+)</div></td>');
55
```

¹⁹ One can use the online tool available at <https://remove-line-numbers.ruurtjan.com/> (Accessed on July 22, 2024) to remove line numbers from the program included here before running it.

```

56          /* Read the HTML line by line */
57          do while (not eof);
58              input html_line $varying32767. reclen;
59              if prxmatch(prx_puf, html_line) > 0 then do;
60                  call prxposn(prx_puf, 1, start, end);
61                  puf_num = substr(html_line, start, end);
62                  end;
63              if prxmatch(prx_meps, html_line) > 0 then do;
64                  call prxposn(prx_meps, 1, start, end);
65                  meps_file = substr(html_line, start, end);
66                  meps_file = prxchange('s/<.+>/', -1, meps_file);      /* remove any stray html tags */
67              end;
68              if prxmatch(prx_data_year, html_line) > 0 then do;
69                  call prxposn(prx_data_year, 1, start, end);
70                  data_year = substr(html_line, start, end);
71                  output year_list_&year.;
72                  end;
73          end;
74      run;
75
76      filename yearresp clear;
77
78      %mend get_year;
79
80      /* Loop through each year of the year_values dataset and call the macro */
81      data _null_;
82          set year_values;
83          call execute("%nrstr(%get_year('||strip(year)||'))");
84      run;
85
86      /* Concatenate all year_list_YEAR datasets */
87      /* Just keep obs with puf_num beginning HC- */
88      data year_list;
89          set year_list_ ;
90          where substr(puf_num, 1, 2) = 'HC';
91      run;
92
93      /* De-dup */
94      proc sort data=year_list nodupkey;
95          by _ALL_;
96      run;
97
98      /* Step 4: Get the zip files for each HC- puf_num. */
99      %macro get_puf(pufnum);
100          %local base_url puf_url;
101          filename pufresp temp;
102
103          /* URL elements */
104          %let base_url = https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?
cboPufNumber=;
105
106          /* Combine the URL elements to construct the PUF url */
107          %let puf_url = &base_url.&pufnum.;
108
109          proc http
110              url="&puf_url."
111              out= pufresp;
112      run;
113
114      data puf_list_%sysfunc(translate(&pufnum, '_', '-')) (keep=puf_num file_format zip_link);
115          length puf_num $15 file_format $150 zip_link $200;
116          infile pufresp length = reclen lrecl = 32767 end=eof;

```

```

117
118     /* regex for the data file name */
119     retain prx_data_file;
120     if _n_ = 1 then prx_data_file = prxparse('/<td width="50%" height="0" class="bottomRightgrayBorder">Data
File.* (.+)</td>');
121
122     /* regex for the zip file download link */
123     retain prx_zip;
124     if _n_ = 1 then prx_zip = prxparse('/<a href="\.\.V(data_files[Vpufs]*V.+\.zip)">ZIP</a>');
125
126     do while (not eof);
127         input html_line $varying32767. reclen;
128         puf_num = "&pufnum.";
129         if prxmatch(prx_data_file, html_line) > 0 then do;
130             call prxposn(prx_data_file, 1, start, end);
131             file_format = substr(html_line, start, end);
132             end;
133
134             if prxmatch(prx_zip, html_line) > 0 then do;
135                 call prxposn(prx_zip, 1, start, end);
136                 zip_link = cats("https://meps.ahrq.gov/", substr(html_line, start, end));
137                 output puf_list_%sysfunc(translate(&pufnum, '_', '-'));
138                 end;
139             end;
140
141     run;
142
143     filename pufresp clear;
144
145 %mend get_puf;
146
147 /* Loop through each puf_num of the year_list dataset and call the macro */
148 data _null_;
149     set year_list;
150     call execute("%nrstr(%get_puf('||strip(puf_num)||'))");
151 run;
152
153 /* Concatenate all puf_list datasets */
154 data puf_list;
155     set puf_list_ ;
156 run;
157
158 /* De-dup */
159 proc sort data=puf_list nodupkey;
160     by _ALL_;
161 run;
162
163 /* Step 5: Merge year_list and puf_list */
164 proc sql;
165     create table meps_zip_links as
166     select y.puf_num, y.meps_file, y.data_year, p.file_format, p.zip_link
167     from year_list as y,
168         puf_list as p
169     where y.puf_num = p.puf_num
170     order by data_year desc, puf_num, file_format;
171 quit;
172
173 /* Format the current date as "YYYY-MM-DD" */
174 %let current_date = %sysfunc(today());
175 %let formatted_date = %sysfunc(putn(&current_date., yymmdd10.));
176
177 /* Step 6: Direct proc report output to excel */

```

```

178 ods listing close;
179 ods excel file = "&path\SAS_Solution_WS_&formatted_date..xlsx"
180   options (sheet_name = 'Sheet1'
181     flow="header,data" row_heights = '15'
182     absolute_column_width='11,11,70,30,55');
183 proc report data=meps_zip_links;
184   column data_year puf_num meps_file file_format zip_link;
185   define puf_num / display ;
186   define meps_file / display;
187   define data_year / display;
188   define file_format / display;
189   define zip_link / display ;
190   compute zip_link ;
191     call define(_col_,"url",zip_link);
192   endcomp;
193 run;
194 ods excel close;
195 ods listing;

```

APPENDIX II (Python program²⁰)

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # **step 1**
5 # - note the python libraries/modules
6
7 # import libraries
8 import pandas as pd
9 import requests
10 from bs4 import BeautifulSoup
11 from urllib.parse import quote
12
13 # **step 2**
14 # - get the list of numerical year options from the main page
15 # - save as list of tuples with year and url
16
17 # main page response
18 main_page = requests.get("https://meps.ahrq.gov/data_stats/download_data_files.jsp")
19 main_soup = BeautifulSoup(main_page.text, "html.parser")
20
21 # get list of data years and links for each year
22 # skip the "All years" (i.e. non-digit options)
23 base_year_url = "https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_results.jsp?cboDataYear="
24 year_url_suffix = "&buttonYearandDataType=Search"
25
26 year_options = main_soup.select_one('select[name="cboDataYear"]').select("option")
27
28 year_url_list = [
29     (x.get("value"), base_year_url + quote(x.get("value"))) + year_url_suffix)
30     for x in year_options
31     if x.get("value").isdigit()
32 ]
33
34 # **step 3**
35 # - use the year_url_list list of tuples to get "HC-" zip file links
36 # - save these in list of dictionaries
37
38 base_hc_url = "https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber="
39 zip_link_list = []
40 for year, year_url in year_url_list:
41     try:
42         response = requests.get(year_url)
43         response.raise_for_status()
44     except:
45         pass
46     dfs = pd.read_html(response.text)
47     hc_df = pd.DataFrame()
48     for df in dfs:
49         if "File(s), Documentation & Codebooks" in df.columns:
50             hc_df = pd.concat([hc_df, df], ignore_index=True)
51             hc_df = hc_df.dropna(subset="PUF no.")
52             hc_df = hc_df.loc[hc_df["PUF no."].str.contains("HC-")]
53             hc_df = hc_df.drop_duplicates(subset="PUF no.")
54             hc_df["hc_url"] = base_hc_url + hc_df["PUF no."]
55     if not hc_df.empty:
56         hc_df.columns = [
57             "puf_num",
```

²⁰ One can use the online tool available at <https://remove-line-numbers.ruurtjan.com/> (Accessed on July 22, 2024) to remove line numbers from the program included here, ensuring the correct indentation level before running it.

```

57         "Files",
58         "Data_Update",
59         "Year",
60         "File_Type",
61         "hc_url",
62     ]
63     for row in hc_df.itertuples():
64         hc_response = requests.get(row.hc_url)
65         hc_response.raise_for_status()
66         hc_soup = BeautifulSoup(hc_response.text, features="lxml")
67         try:
68             meps_file = hc_soup.find(class_="OrangeBox").text
69             for td in hc_soup.find_all("td"):
70                 zip_link_dict = {}
71                 if td.text.startswith("Data File"):
72                     zip_link_dict["data_year"] = row.Year
73                     zip_link_dict["puf_num"] = row.puf_num
74                     zip_link_dict["meps_file"] = meps_file
75                     zip_link_dict["file_format"] = td.text
76                     zip_link_dict[
77                         "zip_link"
78                     ] = "https://meps.ahrq.gov" + td.find_next("a").get(
79                         "href"
80                     ).strip(
81                         ".."
82                     )
83                     zip_link_list.append(zip_link_dict)
84         except:
85             # Ignore any errors and continue processing the next file.
86             pass
87     except requests.exceptions.HTTPError as err:
88         print(err)
89     except requests.exceptions.HTTPError as httperr:
90         print(httperr)
91
92     # **step 4**
93     # - save the list of dictionaries to pandas dataframe
94     # - deduplicate
95     # - clean up meps_file column to remove "PUF no." prefix
96     # - clean up file_format column to remove "Data File" prefix
97     # - sort the dataframe in certain order
98     # - get n rows of the dataframe
99
100     meps_df = pd.DataFrame(zip_link_list)
101     meps_df.drop_duplicates(inplace=True)
102     meps_df["meps_file"] = meps_df["meps_file"].str.split(":", n=1).str[-1]
103     meps_df["file_format"] = meps_df["file_format"].str.split(" ", n=1).str[-1]
104     meps_df = meps_df.sort_values(by=["data_year", "puf_num", "file_format"], ascending=[False, True, True])
105     print(meps_df.head(5))
106
107     # **step 5**
108     # - obtain the current date (timestamp)
109     # - save the dataframe as an excel file
110
111     today = pd.Timestamp("now").strftime("%Y-%m-%d")
112     meps_df.to_excel(f"C:\\SESUG_2023\\Python_Solution_WS_{today}.xlsx", index=False)

```