US Census Current Estimates Weblinks Scrape Report

Abhishek Khatti

Western Governors University

# Contents

[Contents 2](#_Toc20931320)

[Synopsis 3](#_Toc20931321)

[Assumptions and Pre-Requisites 3](#_Toc20931322)

[Structure and Source code of Web Scraper 4](#_Toc20931323)

[Structure and Execution Call for the Program 4](#_Toc20931324)

[Source Code for Web Link Scraping 5](#_Toc20931325)

[Explanation of Code Blocks in the Program 7](#_Toc20931326)

[Import Packages 7](#_Toc20931327)

[OS Package 8](#_Toc20931328)

[Logging Package 8](#_Toc20931329)

[Sys Package 8](#_Toc20931330)

[Unquote, BeautifulSoup and WebDriver Functions 8](#_Toc20931331)

[Web links Extraction 9](#_Toc20931332)

[Scrape HTML code from Web Page 9](#_Toc20931333)

[Scrape URLs from the HTML code 10](#_Toc20931334)

[Data Wrangling and Saving Final Output to CSV File 10](#_Toc20931335)

[Check for File Path Directories 11](#_Toc20931336)

[External HTML page Locator & Remove Possible Duplicates 11](#_Toc20931337)

[Handling External Page & Absolute URLs 11](#_Toc20931338)

[Handling Relative Links 13](#_Toc20931339)

[Write Final Absolute URI list to CSV File 14](#_Toc20931340)

[Scraping HTML Code of Web Page 15](#_Toc20931341)

[Bringing it all together with Main Function 15](#_Toc20931342)

[Execution of the Program & Results 18](#_Toc20931343)

[Acknowledgment 19](#_Toc20931344)

[References 20](#_Toc20931345)

US Census Current Estimates Weblinks Scrape Report

# Synopsis

The data analysts commonly extract large volumes of data from websites and save them locally to analyze phenomenon under investigation. A frequently used scenario is to save all the weblinks in a page for processing later or to automate maintenance tasks on a website, such as checking links or validating HTML code ("Performance Assessment — AAM1"). Python is used to create a program to scrape the weblinks from the HTML code of “U.S. Census Bureau’s Population Estimates”. The list of scraped web links must be unique and absolute. The original HTML code of the website also must be downloaded and saved as a separate file. This report provides an explanation of execution of the Python program and what criteria were used to determine the uniqueness of the Uniform Resource Identifies (URIs).

# Assumptions and Pre-Requisites

There are certain pre-requisites and assumptions the program adheres to. The major assumptions are:

1. All the relative web links assumed to be pointing to the “https://www.census.gov” domain, after checking a sample of 2-3 relative links.
2. The program was built using Python 3.7. For a successful execution of this program, the executing machine would need to be on at this version to avoid failures
3. The program would only run for “http://www.census.gov/programs-surveys/popest.html”, which is the link provided ("Performance Assessment — AAM1").

The pre-requisites for successfully executing this program are:

1. All the necessary packages need to be installed. The list of required packages is provided in subsequent sections.
2. The Web Driver used for this program is exclusively only for the Chrome browser. The executing machine needs to have Chrome Web Driver available for version which is supported by the Selenium Package installed on that machine.

# Structure and Source code of Web Scraper

The “U.S. Census Bureau’s Population Estimates” webpage has a wide range of web links, which either points to other web pages within the same domain or to web pages outside that domain. The Python program “*USCensus-CurrentEst-WebScrape.py*” is used to scrape the weblinks from the “*U.S. Census Bureau’s Population Estimates*” webpage and generate a unique list of URIs from that webpage. The program also converts the relative links to the absolute URIs.

For simplicity, the terms URL (Uniform Resource Locator) and URI (Uniform Resource Identifier) are used interchangeably within the subsequent sections. Although, it needs to be noted that URLs are technically a subset of URIs.

## Structure and Execution Call for the Program

Typically, as a best practice, Python programs follow a proper sequencing of package imports, function definitions and function calls (Weinman, "Python Essential Training", 2018). In adherence to these practices, the below structure was followed when writing the program:

1. Import all required packages
2. Define Custom function and the function body
3. Define the Main function and the function body
4. Call the Main function, along with passing any required arguments

The explanation of each of the components of the program follows the same structure as above.

## Source Code for WebLink Scraping

The complete source code for the web link scraping for python program code is as follows:

import os  
import logging  
import sys  
from urllib.parse import unquote  
from bs4 import BeautifulSoup  
from selenium import webdriver  
  
  
def scrape\_html(src\_url):  
 driver = webdriver.Chrome() # Open Chrome Instance to load the link  
 driver.get(src\_url)  
  
 # Run this script to get the html code after loading all scripts  
 html = driver.execute\_script("return document.documentElement.outerHTML")  
 soup = BeautifulSoup(html, "html.parser")  
  
 # get list of all html anchors  
 anchors = soup.find\_all("a") # print(f"Number of Anchor tags: {len(anchors)}")  
 return anchors, html  
  
  
def scrape\_url(anchors):  
 uris\_set = set() # Use set to avoid duplicates in the array  
  
 for each in anchors:  
 try: # if anchor tag does not have href link, skip tag  
 href = each["href"]  
 except:  
 continue  
 uris\_set.add(href)  
 return uris\_set  
  
  
def write\_to\_files(file\_path, uris\_set):  
 if not os.path.isdir(os.path.dirname(file\_path)):  
 try:  
 os.makedirs(os.path.dirname(file\_path))  
 print(os.path.dirname(file\_path), ": Directory Created")  
 except OSError as excep:  
 print(excep.errno, ": ", excep.strerror)  
  
 abs\_url\_set = set() # Create  
  
 for each in uris\_set:  
 if "http" in unquote(each) or "https" in unquote(each):  
 if "#" in unquote(each):  
 abs\_url\_set.add(str(each).split("#", 1)[0])  
 # Remove duplicate navigation of same URI with http/https tag  
 else:  
 abs\_url\_set.add(str(each).rstrip("/"))  
 # Remove / at the end to avoid duplicate URI scenario  
 else:  
 if "#" in unquote(each):  
 # Remove duplicate navigation of same URI without http/https; avoid blank entries  
 abs\_url\_set.add(str("https://www.census.gov" + str(each).split("#", 1)[0]))  
  
 else:  
 # concatenate full address for relative links  
 abs\_url\_set.add(str("https://www.census.gov" + each).rstrip("/"))  
  
 with open(file\_path, "w+") as writeurls:  
 for each in list(abs\_url\_set):  
 writeurls.write(each)  
 writeurls.write("\n")  
 writeurls.close()  
  
  
def write\_html\_code(html\_path, html\_code):  
 if not os.path.isdir(os.path.dirname(html\_path)):  
 try:  
 os.makedirs(os.path.dirname(html\_path))  
 print(os.path.dirname(html\_path), ": Directory Created")  
 except OSError as excep:  
 print(excep.errno, ": ", excep.strerror)  
  
 with open(html\_path, "w+", encoding="utf-8") as writehtml:  
 writehtml.write(html\_code)  
 writehtml.close()  
  
  
def main(url):  
 logger = logging.getLogger(\_\_name\_\_)  
 log\_fmt = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"  
 logging.basicConfig(level=logging.INFO, format=log\_fmt)  
  
 logger.info("Getting all URIs from US Census Bureau - Current Estimates")  
  
 url\_file\_dir = os.path.join(os.path.curdir, "Extracted Files")  
 url\_file\_path = os.path.join(url\_file\_dir, "USCensus-CurrentEst-ScrapedWebLinks.csv")  
  
 try:  
 anchors, html = scrape\_html(url)  
 urls = scrape\_url(anchors) # Scrape URIs and save to CSV file  
 logger.info("All URIs generated")  
 except EnvironmentError as ex:  
 logger.error("There was an error while scraping for Unique URIs", ex)  
 sys.exit() # Exit script if any issue with the function calls  
  
 logger.info("Writing URIs to the file")  
  
 try:  
 write\_to\_files(url\_file\_path, urls)  
 except EnvironmentError as ex:  
 logger.error("There was an error creating CSV file for Unique URIs", ex)  
 sys.exit()  
  
 logger.info("File with unique URIs generated successfully")  
 html\_file\_path = os.path.join(url\_file\_dir, "USCensus-CurrentEst-HtmlCode.html")  
 logger.info("Writing HTML Code to the file")  
  
 try:  
 write\_html\_code(html\_file\_path, html) # Write the HTML source from the website to HTML file  
 logger.info("File with html code generated successfully")  
 except EnvironmentError as ex:  
 logger.error("There was an error creating HTML file for Current Estimates site", ex)  
 sys.exit()  
  
 logger.info("Script Completed successfully")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main("http://www.census.gov/programs-surveys/popest.html")

# Explanation of Code Blocks in the Program

The following sections explain in detail what each of the code blocks performs and how the end results are achieved.

## Import Packages

There were few Python packages imported to make the writing of code easy. The following sections describe the reasons why each of the package was used.

### OS Package

This package was imported to be able to automatically create the files and directories necessary for the successful output of the program. For example, the target directory was created using the command “os.makedirs(os.path.dirname(file\_path))”.

### Logging Package

This package was imported to be able to capture the required custom logs from the program. This will provide necessary details on the status of the program run and provide necessary details in case of errors. For example, the program updates the log with information on the status of the program run with the command “logger.info("Script Completed successfully")” (Kumar, "Doing Data Science with Python", 2019).

### Sys Package

This package was imported to safely terminate the program in case there are errors while running the program. For example, if there is an exception while generating the CSV file, the command “sys.exit()” will safely terminate the program.

### Unquote, BeautifulSoup and WebDriver Functions

A few specific functions from various packages were imported to perform the necessary tasks of web scraping.

The *unquote* function was imported from urllib.parse package to convert the URIs to string format for easy writing to CSV file.

*BeautifulSoup* was imported from the *bs4* package enable us to scrape the data from the webpages ("Day 25: Web Scraping on Javascript Driven HTML using Python - Part 1", 2017).

The *webdriver* function was imported from the selenium package to successfully launch the website and let the java scripts to be able to successfully load on the web page ("Day 25: Web Scraping on Javascript Driven HTML using Python - Part 1", 2017).

# Weblinks Extraction

The extraction of weblinks is completed in two parts within the program. The first part will extract the HTML code from the webpage and the second part would extract all the web links on the webpage. These links might not be unique and might include the relative links. The program will then address the relative links and potentially duplicate links in subsequent part of code.

## Scrape HTML code from Web Page

The first part extracts the HTML code from the “U.S. Census Bureau’s Population Estimates” web page. This is taken care in “*scrape\_html*” function. This user-defined function launches the Chrome web browser using the Selenium package’s “*webdriver*” function and loads the website with the URL link passed as a parameter to “*scrape\_html*” function.

The function next executes the “*return document.documentElement.outerHTML*” script to load all the java scripts and make the final HTML code available for us to use for scraping ("Day 25: Web Scraping on Javascript Driven HTML using Python - Part 1", 2017). Using this raw HTML code, we create a “*soup*” variable using the BeautifulSoup function, using “*html.parser*” option. The function then searches for all the occurrences of “a” – anchor tags from the soup variable. The function finally returns the raw HTML code and the scraped anchor tags.

The code block which defines the function and executes the above logic is:

def scrape\_html(src\_url):  
 driver = webdriver.Chrome() # Open Chrome Instance to load the link  
 driver.get(src\_url)  
  
 # Run this script to get the html code after loading all scripts  
 html = driver.execute\_script("return document.documentElement.outerHTML")  
 soup = BeautifulSoup(html, "html.parser")  
  
 # get list of all html anchors  
 anchors = soup.find\_all("a") # print(f"Number of Anchor tags: {len(anchors)}")  
 return anchors, html

## Scrape URLs from the HTML code

The second part extracts all the absolute and relative links from scraped anchor tags. This is taken care of in the “*scrape\_url*” user-defined function.

The function first defines a set – “*uris\_set*”, to hold all the unique absolute and relative URLs. The set data structure would contain only the unique set elements and adding the extracted URLs to the set data type object will automatically take care of the duplicate URLs. This is the first step to make sure of the uniqueness of the URLs.

The function then loops through all the anchor tags passed from the *scrape\_html* function. If the current anchor tag – “ *<a>*” has a “*href*” tag in within it, it would write its content to the “*href*” variable, else it would skip that instance of anchor tag. The function then adds the *href* variable to the *uris\_set*. Once the loop is complete with all anchor tags, the function will return the final *uris\_set* structure, along with its unique values.

The code block which defines the function and executes the above logic is:

def scrape\_url(anchors):  
 uris\_set = set() # Use set to avoid duplicates in the array  
  
 for each in anchors:  
 try: # if anchor tag does not have href link, skip tag  
 href = each["href"]  
 except:  
 continue  
 uris\_set.add(href)  
 return uris\_set

# Data Wrangling and Saving Final Output to CSV File

Once all the possible unique URLs (absolute and relative) are scraped, the URLs are then supposed to be transformed to absolute URLs, if they are relative links. In addition, any possible duplicate URLs which might point to the same page are also to be addressed. This is done in “*write\_to\_files*” user-defined function. There are two input parameters for this function – *file\_path* and *uris\_set*. The *file\_path* parameter contains the full path where the final output CSV file is to be saved. The parameter *uris\_set* contains the unique URL set values generated from *scrape\_html* function and all the processing will happen on this set.

## Check for File Path Directories

The first step of this function is to check whether the target directories exist or not. If the directories do not exist, “*os.makedirs(os.path.dirname(file\_path))*” in the “*try*” block of the function will create the required target directories. (Krumelur, "Automatically creating directories with file output", 2017).

The code block which defines the function and executes the above logic is:

def write\_to\_files(file\_path, uris\_set):  
 if not os.path.isdir(os.path.dirname(file\_path)):  
 try:  
 os.makedirs(os.path.dirname(file\_path))  
 print(os.path.dirname(file\_path), ": Directory Created")  
 except OSError as excep:  
 print(excep.errno, ": ", excep.strerror)

## External HTML page Locator & Remove Possible Duplicates

Another set structure is created called “*abs\_url\_set*” to hold the final and unique absolute URLs. After the required directories are created, the *write\_to\_files* function loops through “*each*” element of the *uris\_set* set structure and check a series of conditions.

### Handling External Page & Absolute URLs

The first condition in the loop it checks if the set element from *uris\_set* has “*http*” or “*https*” in the link name (*IF Condition 1)*. If the conditions are met, it would identify two possible conditions. First, if the URI object is an absolute URI and is not a relative link. Second, if the URL is a link for an external HTML page since relative links point to pages from the same domain. To make comparison between a string object and the URI string type easier, the “unquote()” function was used to convert URI string to normal string for comparison (Jaymon, "Transform URL string into normal string in python (%20 to space etc)", 2018).

If the above condition is satisfied, the *write\_to\_files* function then checks to see if the web link has any additional options to load on the webpage, defined after special character “#” (*IF Condition 2)*. For example, the link could have an option to skip the side navigation bar on the web page, defined after character #, resulting in the web link as “http://www.census.gov#skipSideNav”. If we load this weblink to test, it would result in exact same web page as “http://www.census.gov”, which although technically is a different link from “http://www.census.gov#skipSideNav”, it functionally results in a duplicate web link since it points to the same web page. If URI set element currently being processed by the loop has these additional options defined after character #, the string “*split*” function will remove the # character and the entire string after #, resulting in an absolute URI. This generated URI is then added to *abs\_url\_set* structure. If the generated absolute URI is already added to *abs\_url\_set* structure, the set structure will automatically handle that duplicate entry.

In an event the above *IF Condition 2* is not met, it can be inferred that the current URI set element from *uris\_set* processed by the loop does not have a “#” character. Thus, it would execute the “*else*” part of the code *(ELSE for Condition 2)* and it could be concluded that the current URI set element is an absolute URI. This code block simply removes the trailing “/”, if the link has it and then adds it to the *abs\_url\_set* structure. The trailing forward slash is removed from link to avoid potential duplicate web links, such as “http://www.commerce.gov” and “http://www.commerce.gov/”.

### Handling Relative Links

In case the URI set element from *uris\_set* processed by the loop is a relative link, it needs to be handled accordingly. The relative links point to a different web page within the same domain. For example, the relative link “/programs-surveys/popest/library/visualizations.html” in actuality points to “https://www.census.gov/programs-surveys/popest/library/visualizations.html”. In order to convert relative links to absolute URIs, the string “https://www.census.gov” is prefixed to all relative links.

If the values “http” or “https” is not found within the current URI set element of *uris\_set*, it means that the current element is a relative link. The “*else*” part of the code block *(ELSE for Condition 1)*, the same condition is checked to see if the relative link would have any additional options defined within the link separated by the “#” character *(IF Condition 3)*. If the relative link consists the additional options, this code section will remove the entire string after # character, including the # character itself, along with any trailing forward slashes. The absolute URI link is then added to *abs\_url\_set* structure.

In case none of the conditions are met *(ELSE for Condition 3)*, it means that the URI object currently being processed is a relative link without any additional options defined in it. In this case, the relative link is prefixed with the “https://www.census.gov” and the trailing forward slashes are removed, before adding the generated absolute URI to *abs\_url\_set* structure.

The code that executes the above logic within write\_to\_files() function is:

abs\_url\_set = set() # Create  
  
for each in uris\_set:  
 if "http" in unquote(each) or "https" in unquote(each):  
 if "#" in unquote(each):  
 abs\_url\_set.add(str(each).split("#", 1)[0])  
 # Remove duplicate navigation of same URI with http/https tag  
 else:  
 abs\_url\_set.add(str(each).rstrip("/"))  
 # Remove / at the end to avoid duplicate URI scenario  
 else:  
 if "#" in unquote(each):  
 # Remove duplicate navigation of same URI without http/https; avoid blank entries  
 abs\_url\_set.add(str("https://www.census.gov" + str(each).split("#", 1)[0]))  
  
 else:  
 # concatenate full address for relative links  
 abs\_url\_set.add(str("https://www.census.gov" + each).rstrip("/"))

## Write Final Absolute URI list to CSV File

Once all the URI set elements from the *uris\_set* structure are processed within the loop of *write\_to\_files* function, the *abs\_url\_set* structure would have a unique and absolute URI set. This set of unique and absolute URIs will then be written to a CSV file. The name and path of the CSV file are passed from the *main* function, which is explained in the later section.

The *write\_to\_files* function executes the “*open*” function, with the first parameter as “*file\_path*” containing the target location and file name and second parameter as “*w+*”. The *w+* option makes sure if the file is not created, it would create the file first and then open the file in write mode. In case the file is already created, it would first truncate the file and the stream is positioned at the start of the file.

The *write\_to\_files* function then loops through “*each*” of the record from “*abs\_url\_set*”, after it is converted to “*list*” type using *list* function. Each of the records is then written to the CSV file, along with a new line character “*\n*”. Once all the records are written to the CSV file, the file is then closed using the “*close*” function.

The code that executes the above logic within *write\_to\_file*s function is below:

with open(file\_path, "w+") as writeurls:  
 for each in list(abs\_url\_set):  
 writeurls.write(each)  
 writeurls.write("\n")  
 writeurls.close()

# Scraping HTML Code of Web Page

To provide a correct reconciliation of the output file that was generated with the original HTML code of the “Current Estimates” web page, the HTML code needs to be written to an offline copy. This is achieved with “*write\_html\_code*” user-defined function, where the “*html\_path*” is the first parameter which consists of the target file path and file name, and the second parameter “*html\_code*” would contain the actual HTML code that needs to be written to the target file. Both the parameter values are passed from the *main* function.

Similar to *write\_to\_files* function defined earlier, the *write\_html\_code* function also checks if the target directory exists or not. If it does not exist, the function would first create the target directory.

The function then executes the “*open*” function with the target file path as the first parameter and “*w+*” as second parameter. The third optional “*encoding*” parameter is also set to “*utf-8*” to ensure that the file is written with Unicode encoding. The file is then written with the *html\_code* and then is finally closed with the “*close*” function.

The code block which defines the function and executes the above logic is:

def write\_html\_code(html\_path, html\_code):  
 if not os.path.isdir(os.path.dirname(html\_path)):  
 try:  
 os.makedirs(os.path.dirname(html\_path))  
 print(os.path.dirname(html\_path), ": Directory Created")  
 except OSError as excep:  
 print(excep.errno, ": ", excep.strerror)  
  
 with open(html\_path, "w+", encoding="utf-8") as writehtml:  
 writehtml.write(html\_code)  
 writehtml.close()

# Bringing it all together with Main Function

All the user-defined functions discussed earlier, *scrape\_html*, *scrape\_url*, *write\_to\_files,* and *write\_html\_code*, are currently stand-alone functions and will need to be called in a proper sequence, along with passing all the correct parameters to each of the functions. The “*main*” function does this job perfectly.

The *main* function has only one parameter, “*url*”, which will hold the web link for the web page where all the other web links need to be scraped from.

The *main* function first sets the logger, log format and the configuration of the logger to display the information type logs. By default, only warning and error type logs are displayed in the console for the logger (Kumar, "Doing Data Science with Python", 2019).

Once the logger is set correctly, the *main* function then generates the target file directory using the “*os.path*” functions and saves it under the “*url\_file\_dir*” variable. This variable would be later used to generate target file paths by concatenating the file names. The function then generates the target file path for CSV file to save the final, unique and absolute URI list. The file path is generated by concatenating the *url\_file\_dir* and file name string, “USCensus-CurrentEst-ScrapedWebLinks.csv”.

The *scrape\_html* and *scrape\_url* functions are then called within the *try* block. The “*url*” parameter passed as the main function argument, is then passed to *scrape\_html* function and it returns two output parameters – *anchors* and *html*. The “*anchors*” variable consists of all the anchor tags scraped from the web page and the “*html*” variable consists of the actual HTML code scraped from the “Current Estimates” web page.

The *anchors* variable is then passed to *scrape\_url* function. The function then creates the first round of unique URIs from *anchors* list, including the absolute URIs and relative links. The output of this function is saved in “*urls*” variable.

If any environment level exceptions occur while running any of these user-defined functions, the except block will handle that exception to display the appropriate messages and exit out of the program safely.

Once the first set of URLs is scraped, the main function then passes the CSV file path and *urls* set to *write\_to\_files* function. This function will further cleanse the URL list from the “*urls*” variable and write the final, unique and absolute URI list to CSV file. This file will be saved at the target file path, which is passed as the first parameter to the *write\_to\_files* function. Again, if there are any environment level exceptions occur while running this function, the except block will handle that exception to display the appropriate messages and exit out of the program safely.

Once the CSV file has been successfully generated, only then the HTML code of the Current Estimates web page is written to the local HTML file. The target file path is first set in the “*html\_file\_path*” string variable, along with the target file name. This is the first parameter passed to *write\_html\_code* function along with “*html*” variable generated earlier from *scrape\_html* function. The *write\_html\_code* function will then write the code from “*html*” variable to the offline file. This function call is also within *try…except* block, which means any environment level exceptions are caught and program is exited safely.

The code block which defines the function and executes the above logic is:

def main(url):  
 logger = logging.getLogger(\_\_name\_\_)  
 log\_fmt = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"  
 logging.basicConfig(level=logging.INFO, format=log\_fmt)  
  
 logger.info("Getting all URIs from US Census Bureau - Current Estimates")  
  
 url\_file\_dir = os.path.join(os.path.curdir, "Extracted Files")  
 url\_file\_path = os.path.join(url\_file\_dir, "USCensus-CurrentEst-ScrapedWebLinks.csv")  
  
 try:  
 anchors, html = scrape\_html(url)  
 urls = scrape\_url(anchors) # Scrape URIs and save to CSV file  
 logger.info("All URIs generated")  
 except EnvironmentError as ex:  
 logger.error("There was an error while scraping for Unique URIs", ex)  
 sys.exit() # Exit script if any issue with the function calls  
  
 logger.info("Writing URIs to the file")  
  
 try:  
 write\_to\_files(url\_file\_path, urls)  
 except EnvironmentError as ex:  
 logger.error("There was an error creating CSV file for Unique URIs", ex)  
 sys.exit()  
  
 logger.info("File with unique URIs generated successfully")  
 html\_file\_path = os.path.join(url\_file\_dir, "USCensus-CurrentEst-HtmlCode.html")  
 logger.info("Writing HTML Code to the file")  
  
 try:  
 write\_html\_code(html\_file\_path, html) # Write the HTML source from the website to HTML file  
 logger.info("File with html code generated successfully")  
 except EnvironmentError as ex:  
 logger.error("There was an error creating HTML file for Current Estimates site", ex)  
 sys.exit()  
  
 logger.info("Script Completed successfully")

The *main* function is called using a standard and a very common line of code, where we check if the system variable “*\_\_name\_\_*” is equal to “*\_\_main\_\_*”. If this condition is satisfied, the program then calls the *main* function along with any of the required arguments that need to be passed. In this case, the full web link URL for “Current Estimates” is passed as an argument. The code block that runs this logic is:

if \_\_name\_\_ == "\_\_main\_\_":  
 main("http://www.census.gov/programs-surveys/popest.html")

Once both the files are generated successfully, the program will then complete, with all the log information updated and displayed successfully.

# Execution of the Program & Results

The program can be executed from the Windows Command Prompt if the Python interpreter is installed and the System Path is set in Environment Variables. It can also be run through the Git Bash Terminal or through any of the IDE Editors. In this instance, the program is executed using the Git Bash Terminal, from the folder which consists the Python program.

The program run is successful, and the required files are generated in the correct folder.

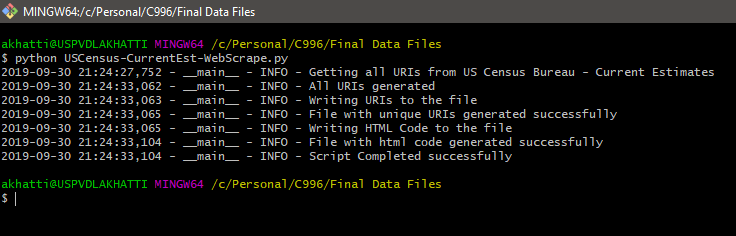


Figure 1. Execution Log of Web Scraping Program

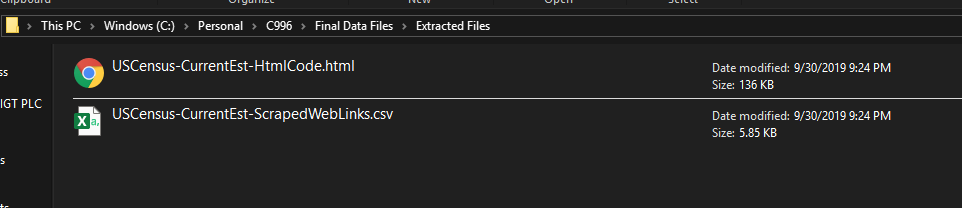


Figure 2. Output Files & File Location

# Acknowledgment

The Python code is written in a free version of PyCharm IDE by Jet Brains. The document template for this report was provided by Writing Center team from Student Success Centers, with additional details referred to APA style formatting from WGU Knowledge Center (WGU Knowledge Center, solene, "APA Document Formatting", 2018).

All the sources listed under the “References” section of this report were used to refer details and instructions to successfully complete this analysis and report.

Finally, sincere gratitude to “Dr. Emelda Ntinglet” for her exceptional guidance, support, and encouragement throughout this course.

# References

W. (n.d.). PERFORMANCE ASSESSMENT — AAM1. Retrieved from https://tasks.wgu.edu/student/001101075/course/14380005/task/1077/overview

Weinman, B. (2018, January 18). Python Essential Training. Retrieved from https://www.linkedin.com/learning/me?u=2045532.

Day 25: Web Scraping on Javascript Driven HTML using Python - Part 1. (2017, November 14). Retrieved from https://youtu.be/vcnomT0CP0Y.

Kumar, A. (2019, July 29). Doing Data Science with Python. Retrieved from https://app.pluralsight.com/library/courses/python-data-science/table-of-contents.

Krumelur. (2017, May 27). Automatically creating directories with file output. Retrieved from https://stackoverflow.com/questions/12517451/automatically-creating-directories-with-file-output.

Jaymon. (2018, January 3). Transform URL string into normal string in python ( to space etc). Retrieved from https://stackoverflow.com/questions/11768070/transform-url-string-into-normal-string-in-python-20-to-space-etc.