Web Scraping and Data Download

Prepared by
Mohamed Abdelkader[1], Jorge Bravo[1], Marouane Temimi[1], and Jibin Joseph[2]
[1]Civil, Environmental, and Ocean Engineering Department, Stevens Institute of Technology
[2]School of Civil Engineering, Purdue University
mabdelka@stevens.edu

FAIR Science in Climate

Objective

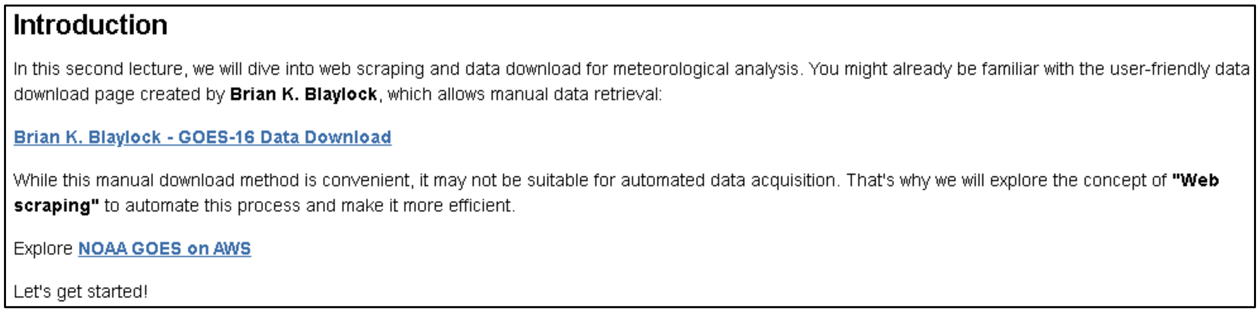
The objective is to introduce students how to use Python for web scraping and automated data downloads, specifically for gathering meteorological data from the GOES-16 satellite. The lecture will cover the use of Python libraries such as datetime, requests, BeautifulSoup, and boto3 to automate the process of constructing URLs, parsing HTML content, and downloading satellite data files. This will enable students to efficiently automate data retrieval for remote sensing applications.

Resources

Guide to GOES-R Series Data

Instructions

1. Access Lecture_2 folder and initiate "Web Scraping and Data Download" notebook.
2. Review the introductory overview of Python's capabilities in remote sensing applications to understand the context and relevance of programming in this field.
3. Familiarize yourself with manual data retrieval from Brian K. Blaylock's GOES-16 Data Download page to understand the basics of data acquisition. Explore the NOAA GOES data available on AWS to grasp the variety of meteorological data accessible for analysis.



Introduction

In this second lecture, we will dive into web scraping and data download for meteorological analysis. You might already be familiar with the user-friendly data download page created by **Brian K. Blaylock**, which allows manual data retrieval:

Brian K. Blaylock - GOES-16 Data Download

While this manual download method is convenient, it may not be suitable for automated data acquisition. That's why we will explore the concept of **"Web scraping"** to automate this process and make it more efficient.

Explore **NOAA GOES on AWS**

Let's get started!

4. Import the `datetime` module to handle date and time in your script.

```
# Importing the datetime module to work with date and time information.
from datetime import datetime
```

The basic date and time types could be obtained using the "datetime" Library https://docs.python.org/3/library/datetime.html

5. Create and display a `datetime` object for the current UTC time to practice working with date and time information.

```
# Create a datetime object representing the current UTC time.
dnow = datetime.utcnow()
```

```
# Display current date
dnow
```

6. *Print a message with the current UTC time to learn how to combine text and datetime objects in output.*

```
# Display a message along with the current UTC time.
print("The current UTC time is:", dnow)
```

7. *Define a dictionary named `dini` with parameters for constructing a data download URL, including data source, satellite, spatial domain, product type, and current time.*

```
# Define a dictionary 'dini' containing parameters for data download URL construction.
dini = {
    'src': 'aws',                      # Data source
    'sat': '16',                       # Satellite (GOES-16)
    'str': 'F',                        # Spatial domain (e.g., 'F' for full disk)
    'prd': 'ABI-L1b-Rad',              # Product type (e.g., ABI-L1b-Rad for radiance data)
    'tme': dnow                        # Date and time (current UTC time)
}
```

8. *Construct a URL string using the `dini` dictionary parameters to automate the construction of a data download link, **paying attention to formatting dates and hours properly**.*

```
# Construct the URL for data download using the provided parameters.
url = (
    'https://home.chpc.utah.edu'
    '/~u0553130/Brian_Blaylock/cgi-bin/goes16_download.cgi?'
    'source={src}&'
    'satellite=noaa-goes{sat}&'
    'domain={str}&'
    'product={prd}&'
    'date={tme:%Y-%m-%d}&'             # Format the date as 'YYYY-MM-DD'
    'hour={tme:%H}'                   # Format the hour as 'HH'
)
```

9. *Print the constructed data download URL by applying the parameters from the `dini` dictionary, to practice string formatting in Python.*

```
# Print the constructed URL with parameter values applied using string formatting.
print("Constructed URL with parameter values:")
print(url.format(**dini))
```

10. *Import the `requests` library to enable making HTTP requests. Following, import `BeautifulSoup` from the `bs4` library for HTML parsing, and `minidom` from `xml.dom` for XML handling.*

```
# Import the 'requests' library for making HTTP requests.
import requests

# Import 'BeautifulSoup' from the 'bs4' library for web scraping and parsing HTML.
from bs4 import BeautifulSoup

# Import 'minidom' from 'xml.dom' for working with XML data.
from xml.dom import minidom
```

11. *Use the formatted URL to make an HTTP GET request and store the response. Parse the response using `BeautifulSoup` to find HTML elements for data download times, displaying the results.*

```
# Construct the complete URL with parameter values applied.
urit = url.format(**dini)

# Make an HTTP GET request to the constructed URL.
response = requests.get(urit)

# Parse the HTML content of the response using BeautifulSoup.
dates = BeautifulSoup(response.text, "html.parser")

# Find all elements with class 'mybtn-group' in the parsed HTML.
alltimesxml = dates.findAll('div', 'mybtn-group')

# Display the scraped HTML content stored in the 'alltimesxml' variable.
alltimesxml
```

12. *Initialize a list to store data links for download and determine the most recent file by comparing timestamps.*

```
# Initialize an empty list to store the data to be downloaded.
ls2down = []

# Initialize 'nps' to -1 as an initial value.
nps = -1

# Iterate through the elements in 'alltimesxml'.
for i in range(len(alltimesxml)):
    # Find all 'a' tags within the current element.
    tags = alltimesxml[i].find_all('a')

    # Initialize an empty list to store time differences.
    nwdts = []

    # Iterate through the 'a' tags.
    for j in range(len(tags)):
        # Extract the date and time information from the 'href' attribute.
        dts = tags[j].attrs['href'].split('/',)[-1]

        # Split the date and time string to extract the timestamp.
        lsfst = dts.split('_')
        dtstr = lsfst[-3]

        # Convert the timestamp to a datetime object.
        ndnow = datetime.strptime(dtstr, 's%Y%j%H%M%S%f')

        # Calculate the time difference in minutes.
        tmdf = ndnow.minute - dnow.minute

        # Append the absolute time difference to the list.
        nwdts.append(abs(tmdf))

    # Find the index with the minimum time difference.
    id1 = min(nwdts)

    # If 'nps' is 0 and the minimum time difference is less than 'dtm', update 'nps'.
    if nps == 0 and id1 < dtm:
        nps = nwdts.index(id1)

    # Extract the filename and button text from the 'a' tag.
    lsfst = tags[nps].attrs['href'].split('/',)[-1]
    print(lsfst + ' - ' + tags[j].button.text)

    # Append the filename to the 'ls2down' list.
    ls2down.append(lsfst)
```

13. *Generate a list of URLs for downloading the identified files using the base URL and the filenames from the previous step.*

```python
# Check if the 'str' key in the dini dictionary contains 'M' (indicating a specific domain)
if 'M' in dini['str']:
    # If 'M' is found, update the 'str' key in the dini dictionary to 'M' for consistency
    dini.update({'str':'M'})

# Construct the base URL for downloading the data using formatted string.
# This URL includes placeholders for satellite (sat), product (prd), and time (tme) parameters,
# which are filled in from the dini dictionary.
url_base = 'https://noaa-goes{sat}.s3.amazonaws.com/{prd}{str}/{tme:%Y}/{tme:%j}/{tme:%H}/'.format(**dini)

# Initialize an empty list to store the complete URLs for downloading the data files
urls2dwn = []

# Iterate over each item in the list of file identifiers (ls2down)
for urli in ls2down:
    # Uncomment the next line to print each constructed URL before adding it to the list (for debugging)
    # print(f'{url_base}{urli}')

    # Append the full URL for each file to the urls2dwn list. This URL is constructed by combining
    # the base URL with the specific file identifier (urli), allowing for direct access to each file.
    urls2dwn.append(f'{url_base}{urli}')
```

14. *Prompt for a download folder path, download a specified file using its URL, and save it to the specified location, checking for successful download.*

```python
import requests
import os

# Prompt the user to enter the download folder path.
download_folder = input("Enter the path to the download folder: ")

# URL of the file to download.
urld = 'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadM/2022/306/17/OR_ABI-L1b-RadM1-M6C13_G16_s20223061734250_e20223061

# Send an HTTP GET request to the URL.
response = requests.get(urld)

# Check if the response is successful (status code 200).
if response.status_code == 200:
    # Extract the filename from the URL.
    filename = os.path.basename(urld)

    # Construct the complete path to save the file in the chosen download folder.
    file_path = os.path.join(download_folder, filename)

    # Write the content to the file in binary mode.
    with open(file_path, "wb") as file:
        file.write(response.content)

    print(f"File '{filename}' downloaded and saved to '{download_folder}'.")
    # Press enter if you want to save the file in the current directory
else:
    print("Failed to download the file. Check the URL or your internet connection.")
```

15. *Print the list of URLs prepared for downloading to display them.*

```python
# Display the URLs prepared for downloading
print("List of URLs to download:")
for index, url in enumerate(urls2dwn, start=1):
    print(f"{index}. {url}")
```

16. *Prompt for a download folder and download a file from a URL, checking for success and handling errors appropriately.*

```python
import requests
import os

try:
    # Prompt the user to enter the download folder path.
    download_folder = input("Enter the path to the download folder: ")

    # Ensure the download folder exists.
    if not os.path.isdir(download_folder):
        print(f"Creating download folder at '{download_folder}'.")
        os.makedirs(download_folder, exist_ok=True)

    # URL of the file to download.
    urld = 'https://noaa-goes16.s3.amazonaws.com/ABI-L1b-RadM/2022/306/17/OR_ABI-L1b-RadM1-M6C13_G16_s20223061734250_e20223

    print("Attempting to download the file...")
    # Send an HTTP GET request to the URL.
    response = requests.get(urld, stream=True)

    # Check if the response is successful (status code 200).
    if response.status_code == 200:
        # Extract the filename from the URL.
        filename = os.path.basename(urld)

        # Construct the complete path to save the file in the chosen download folder.
        file_path = os.path.join(download_folder, filename)

        # Write the content to the file in binary mode.
        with open(file_path, "wb") as file:
            for chunk in response.iter_content(chunk_size=8192):
                file.write(chunk)

        print(f"File '{filename}' downloaded and saved to '{download_folder}'.")
    else:
        print(f"Failed to download the file. Server responded with status code: {response.status_code}. Check the URL or you
except Exception as e:
    print(f"An error occurred: {e}")
```

17. *For downloading multiple files, ensure the target directory exists, then iterate through a list of URLs, downloading each file and saving it in the specified directory. Provide feedback on the status of each download.*

```python
import requests
import os

# Specify the local directory where you want to save the files.
local_directory = input("Enter the path to the download folder: ")

# Ensure that the local directory exists; create it if it doesn't.
os.makedirs(local_directory, exist_ok=True)

# Iterate through the URLs and download files.
for urld in urls2dwn:
    # Extract the filename from the URL.
    ntw = urld.split('/')[-1]

    # Construct the complete path to save the file in the local directory.
    file_path = os.path.join(local_directory, ntw)

    # Send an HTTP GET request to the URL.
    resp = requests.get(urld)

    # Check if the response is successful (status code 200).
    if resp.status_code == 200:
        # Write the content to the file in binary mode.
        with open(file_path, "wb") as file:
            file.write(resp.content)
        print(f"File '{ntw}' downloaded and saved to '{local_directory}'.")
    else:
        print(f"Failed to download '{ntw}' from the URL: {urld}")

print("Download process completed.")
```

18. *Import AWS SDK and configure the client for unsigned requests to access public AWS resources without credentials.*

```python
# Import necessary AWS SDK and configuration modules.
import boto3
from botocore import UNSIGNED
from botocore.client import Config
```

19. *Specify the S3 bucket name, the path to the remote file, and the name for the file once downloaded.*

```python
# Select the AWS S3 bucket name, remote file path, and local destination file name.
s3_bucket = 'noaa-goes16'
bucket_file = 'ABI-L2-MCMIPF/2022/273/03/OR_ABI-L2-MCMIPF-M6_G16_s20222730350207_e20222730359521_c20222730400027.nc'
local_file = 'OR_ABI-L2-MCMIPF-M6_G16_s20222730350207_e20222730359521_c20222730400027.nc'
```

20. *Use the Boto3 client to connect to an S3 bucket without signing requests, then download a specified file to a local path.*

```python
# Connect to the AWS S3 bucket using the Boto3 client.
s3 = boto3.client('s3', config=Config(signature_version=UNSIGNED))
```

```python
# Download the file from the AWS S3 bucket to the local destination.
s3.download_file(s3_bucket, bucket_file, local_file)
```

Ok, you have completed the tutorial!

Turn-in

Homework Assignment: Download GOES Data for Hurricane Ida Landfall

Instructions:
Your task is to download satellite observations from the GOES satellite during the landfall of Hurricane Ida. You will use Python and the skills learned in the "Web Scraping and Data Download" lecture to automatically retrieve data for this significant weather event.

1. Research Hurricane Ida: Identify the exact date and time of Hurricane Ida's landfall. This information will be crucial for constructing your data download queries.

2. Determine the Data You Need:
   - Using the information gathered about Hurricane Ida, decide which GOES satellite data would be most relevant for observing the hurricane. Consider factors such as the type of imagery or data product (e.g., visible, infrared) and the spatial coverage (e.g., CONUS, full disk).

3. Construct the Data Download URL:
   - Based on your selected data type and the timing of Hurricane Ida's landfall, construct a URL to query the GOES data archive. You may use the template provided in the lecture materials, adjusting parameters as necessary to target the specific observations you're interested in.

5. Write a Python Script to Download the Data:
   - Utilize the `requests` library to make an HTTP GET request to your constructed URL.
   - Use `BeautifulSoup` to parse the response and identify the available data files.
   - Download the selected data files to your local machine, using the `boto3` library if accessing AWS storage. Ensure your script handles errors and provides informative messages about the download process.

6. Document Your Process:
   - In a text file or as comments within your Python script, briefly describe the steps you took to select and download the data. Include any challenges you faced and how you overcame them.

7. Submit Your Work:
   - Submit your Python script and documentation. Ensure your script is well-commented, explaining each part of the process for clarity.