Jal Jeevan Mission Water Quality Data - Python Code

July 27, 2024

Har Ghar Jal
Jal Jeevan Mission

Automation using Python

This document provide overview of python code of JJM Data collection and Pre-processing. Their complete codes and codes for Ensuring no missing files and Fuzzy Matching are in thier respective Jupyter notebook. Link to access it.

Code for JJM Data collection

Step 1: Installing Selenium Importing libraries and providing Selenium Webdriver with its location. To be taken care that Webdriver and Chrome Version should be the same.

Code 1.1:

```
pip install selenium --quiet
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException
import time

def get_driver():
    service = Service("chromedriver.exe")
    driver = webdriver.Chrome(service=service)
    return driver
```

Step 2: Selenium Script to Download Files

- Start by selecting the year on the Labtestingdata report on the ejal sakti website. For example, choose the year 2023-2024.
- Iterate through each state using an outer loop.
  - For each state:
    - Select the state and click the "Show" button.
    - The page updates to display all districts in the selected state.
    - Locate the third "Total" column. This column contains clickable buttons. Identify the relevant cells in this column:
      - The cells are tagged with "TD". Start with the 6th "TD" tag.
      - Check every 16th "TD" tag thereafter (e.g., 6th, 22nd, 38th, etc.).
      - Verify if these "TD" tags contain an 'href' attribute, ie they are clickable. Store the positions of clickable cells in an "Order" list.
    - Use the "Order" list to iterate over the cells with links.
    - For each clickable cell:
      - Click the link to open the city page.
      - Download the Excel file by clicking the XLS icon.
      - Return to the previous page.
    - Repeat this process for all cities with clickable links in the third "Total" column.

Code 1.2:

```
JJM = "https://ejalshakti.gov.in/WQMIS/Report/Labtestingdata"
driver = get_driver()
driver.get(JJM)
dropdownbox = driver.find_elements(By.TAG_NAME, 'option')
dropdownbox[2].click()
time.sleep(2)

for i in range(5,41):
    # The State Dropdowns are 5th to 40ths.
    dropdownbox = driver.find_elements(By.TAG_NAME, 'option')
    dropdownbox[i].click()
    time.sleep(2)

    # click on show button
    select = driver.find_element(By.XPATH, '/html/body/div[1]/
    div[2]/div/div/section[2]/div/div[1]/div/div[8]/div/div/a')
    select.click()
    time.sleep(6)

    # No of citites with clickable third Total column
    td_divs = driver.find_elements(By.TAG_NAME, 'td')
    order = []
    if len(td_divs) > 0:
        j = 6
        x = 1
```

Code 1.3:

```
        while j <= len(td_divs):
            td = td_divs[j]
            try:

                # X path is of Xls download icon on left side
                url = td.find_element(By.CLASS_NAME,
                'txtnumber a').get_attribute('href')
                order.append(x)

            except NoSuchElementException:
                print("Element with class
                'txtnumber a' not found in td. Skipping...")
            j = j + 16
            x = x + 1

    for z in order:

        select_link = driver.find_element(By.XPATH,
        f'/html/body/div[1]/div[2]/div/div/section[2]
        /div/div[3]/div/table[1]/tbody/tr[{z}]/td[7]/a')
        select_link.click()
        time.sleep(2)

        # Need to add if statment to deal with issue
        # of No record available
        # When number of smaple in the district are high
        # then the download link is in left side.
```

Code 1.4:

```
        try:
            # X path of Xls download icon on left side

            download_link = driver.find_element(By.XPATH,
            '/html/body/div[1]/div[2]/div/div/section[2]
            /div/div[1]/div[1]/a/i')
            download_link.click()
            time.sleep(5)

        except NoSuchElementException:
            try:
                download_link = driver.find_element(By.XPATH,
                '/html/body/div[1]/div[2]/div/div/section[2]
                /div/div[1]/div[1]/a/i')
                download_link.click()
                time.sleep(5)
                print("Download found right side")

            except NoSuchElementException:
                print("Download option Not Found. Skipping...")

        back_link = driver.find_element(By.XPATH,
        '/html/body/div[1]/div[2]/div/
        div/section[1]/div/div[1]/h4/a')
        back_link.click()
        time.sleep(2)
```

Converting Xls to CSV

The downloaded files are of html format with .xls extension. To use it for any analysis we need to convert it into .csv format and arrange them in District and State folders. Following code performs that process

Code 1.5:

```
from bs4 import BeautifulSoup
import csv
for ele in file_paths:
    ele  = ele.replace("\\", "/")
    with open(ele, 'r', encoding='utf-8') as html_file:
        html_content = html_file.read()
        soup = BeautifulSoup(html_content, 'html.parser')

        # Find the district name
        district_name = ""
        table = soup.find('table')

        if table:
            state_name = table.find_all('tr')[1]
                        .find_all('td')[1].get_text()
            district_name = table.find_all('tr')[1]
                            .find_all('td')[2].get_text()
            sample_collection_date = table.find_all('tr')[1]
                                    .find_all('td')[9].get_text()
            year = sample_collection_date.split('/')[-1]
```

Code 1.6:

```
    # Create CSV file with district name as filename
    if district_name:
        csv_filename = f"{state_name.strip()}_
                        {district_name.strip()}_{year}.csv"
        with open(csv_filename, 'w', newline='',
                  encoding='utf-8') as csvfile:

            writer = csv.writer(csvfile)

            # Write header row
            header_row = [header.get_text() for header
                          in table.find_all('th')]

            writer.writerow(header_row)

            # Write data rows
            for row in table.find_all('tr')[1:]:
                data_row = [data.get_text()
                            for data in row.find_all('td')]
                writer.writerow(data_row)
```

For NRDWP code has few more changes but mostly similar

Preprocessing Code

As explained in Preprocessing document we create two new columns from existing pollutants column with uniform pollutants namings and structure.

Code Overview

- **Importing Libraries:** Importing `pandas`, `re`, and `os` libraries.
- **Function:** `merge_non_nan_values(row)`: Merges non-NaN values from `AbovePMandatory`, `BelowPMandatory`, `AbovePEmerging`, and `BelowPEmerging` into a single string, separated by commas.
- **Function:** `reformat_contaminants(contaminant_string)`: Uses a regular expression to find contaminants and their values in the format `Name[Value Unit]`, reformats them into `Name:Value` format, and joins them with commas.
- **Function:** `clean_contaminants(contaminants)`: Splits the contaminants string by commas, attempts to split each pair into pollutant and value, checks if the pollutant matches any in a predefined dictionary (`standard_names`), replaces the pollutant with its standard name if a match is found, and joins the cleaned contaminants back into a string, separated by commas.
- **Function:** `check_contaminants(row)`: Splits the `Contaminents` string by commas, for each contaminant, splits into name and value, converts the value to a float and checks if it exceeds predefined permissible limits (`permissible_limit`), and joins the exceeded contaminants back into a string, separated by commas.

- **Processing CSV Files:** Iterates through each CSV file in `nrdwp_csv_files`, reads the CSV file into a DataFrame (`df`), merges non-NaN values into the `Contaminents` column using `merge_non_nan_values`, applies `reformat_contaminants` to reformat the `Contaminents` column, applies `clean_contaminants` to standardize the `Contaminents` column, and checks for contaminants exceeding permissible limits using `check_contaminants`, storing the result in a new column `Exceeded_Contaminants`.

- **File Path Extraction and Directory Creation:** Extracts the year range and state name from the file path using a regular expression, constructs a new folder path based on the extracted year range and state name, and creates the new directory if it doesn't exist.

- **Saving Processed Data:** Constructs the new file path within the newly created directory and saves the updated DataFrame to the new file path as a CSV file.

Code 1.7:

```
import re
import os
def merge_non_nan_values(row):
    values = [str(row[col]) for col in ['AbovePMandatory',
              'BelowPMandatory', 'AbovePEmerging',
              'BelowPEmerging'] if pd.notna(row[col])]
    return ','.join(values)
def reformat_contaminants(contaminant_string):
    pattern = re.compile(r'([^,]+)\[(\d+\.\d+)
                        \s*\w+/[a-zA-Z0-9]+\]')
    matches = pattern.findall(contaminant_string)
    reformatted_list = [f"{match[0]}:{match[1]}"
                        for match in matches]
    return ','.join(reformatted_list)
def clean_contaminants(contaminants):
    new_contaminants = []
    pairs = contaminants.split(',')
    for pair in pairs:
        try:
            pollutant, value = pair.split(':')
            for std_name, alt_names in standard_names.items():
                if any(alt in pollutant for alt in alt_names):
                    new_contaminants.append
                    (f"{std_name} : {value}")
                    break  # Stop once the pollutant is found
        except ValueError:
            continue
    return ', '.join(new_contaminants)
```

Code 1.8:

```
def check_contaminants(row):
    contaminants = row['Contaminents'].split(',')
    exceeded_limits = []
    for cont in contaminants:
        try:
            cont_name, cont_value = cont.split(':')
            cont_value = float(cont_value)
            if cont_name.strip() in permissible_limit
            and cont_value > permissible_limit[cont_name.strip()]:

                exceeded_limits.append
                (f"{cont_name.strip()}:{cont_value}")

        except ValueError:
            # If splitting the contaminant string fails,
            # skip to the next contaminant
            continue

    return ','.join(exceeded_limits) if exceeded_limits else None

for csv in nrdwp_csv_files:
    df = pd.read_csv(csv)
    df['Contaminents'] = df.apply(merge_non_nan_values, axis=1)
    df['Contaminents'] = df['Contaminents']
                            .apply(reformat_contaminants)
    df['Contaminents'] = df['Contaminents']
                            .apply(clean_contaminants)
```

Code 1.9:

```
    df['Exceeded_Contaminants'] = df.apply(check_contaminants)

    # Extract year and state name from the file path
    match = re.search(r'(\d{4})-(\d{4})\\CSV \d{4}-\d{4}
                    \\([^\\]+)\\([^\\]+)_\d{4}\.csv', csv)

    if match:
        year_range = match.group(1) + "-" + match.group(2)
        state_name = match.group(3)

        # Construct the new file path
        new_folder = f"JJM 2009 - 2024 Water Quality Data
                    \\JJM {year_range}\\Processed CSV
                    {year_range.split('-')[1]}\\{state_name}"

        os.makedirs(new_folder, exist_ok=True)
        new_file_path = os.path.join(new_folder,
                        os.path.basename(csv))

        # Save the updated DataFrame
        df.to_csv(new_file_path, index=False)
```

This code Preprocesses JJM scraped Data, For NRDWP data, code has few more changes but is mostly similar.