Coding Sample

Bishmay Barik

January 1, 2026

1 Project Overview: lipi-swap

This Python script was developed to scrape data from an Indian administrative database, where names of entities like Panchayats and Blocks are often written in Hindi (Devanagari script). The script uses web scraping techniques to collect data and leverages the `unidecode` library to transliterate Hindi names into English, ensuring compatibility when saved to a CSV file.

1.1 Purpose and Functionality

This script is built to scrape hierarchical data from Indian datasets (e.g., states, districts, blocks, and panchayats) using `requests` and `BeautifulSoup`. It handles non-Roman scripts like Devanagari by converting them to English equivalents with `unidecode`. The scraped data, including monthly figures, is organized into a structured CSV file using `pandas`. English names remain unchanged, while Hindi names are transliterated for readability and compatibility.

1.2 Why Transliteration?

When dealing with datasets containing Devanagari script (e.g., Hindi names), saving them directly to a CSV can result in encoding issues or unreadable characters on systems not configured for non-Roman scripts. Transliteration solves this by converting Hindi characters to their closest English (Roman) equivalents, making the data portable and usable across different platforms.

1.3 Installation

To run **lipi-swap**, you'll need the following Python libraries:

- `requests` - For making HTTP requests
- `beautifulsoup4` - For parsing HTML content
- `pandas` - For handling data and CSV output
- `unidecode` - For transliterating Hindi to English

Install them using pip:

```
pip install requests beautifulsoup4 pandas unidecode
```

2 Refactored Code Following Rigorous Standards

I have refactored this code to adhere to rigorous coding standards, including:

- **Extensive commenting:** One comment for every line of code (minimum 1:1 ratio)

- **Proper naming conventions:** Using `df_` prefix for DataFrames, `_list` suffix for lists, `_path` suffix for file paths
- **Clear function documentation:** Comprehensive docstrings with Args, Returns, and Example sections
- **Verbose parameter:** Added to control debug output
- **Clear error messages:** Informative messages for debugging
- **Interpretability over brevity:** Code prioritizes clarity and maintainability

2.1 lipi-swap.py (Refactored)

```python
"""
lipi-swap: Web scraper for Indian administrative database with Hindi
    transliteration

This script scrapes hierarchical data (states, districts, blocks, panchayats)
    from
Indian administrative databases and transliterates Devanagari (Hindi) names to
    Roman
script for CSV compatibility.
"""

import requests
from bs4 import BeautifulSoup
import pandas as pd
from urllib.parse import urljoin, urlparse, parse_qs
import time
from unidecode import unidecode

# define base URL for constructing full URLs
BASE_URL_PATH = "put/your/base/url/here"

# define list of state URLs (should be provided externally)
state_url_list = [
    # Add state URLs dynamically
]


def get_soup(url, verbose=True):
    """
    Fetch and parse a webpage into a BeautifulSoup object.

    Args:
        url (str): The URL to fetch
        verbose (bool): If True, print status messages. Default is True.

    Returns:
        BeautifulSoup: Parsed HTML content, or None if request fails

    Example:
        >>> soup = get_soup("https://example.com")
        >>> soup is not None
        True
    """
    try:
        # send GET request to the URL
        response = requests.get(url)

        # raise exception if status code indicates error
```

```
        response.raise_for_status()

        # add delay to prevent server overload
        time.sleep(1)

        # parse response text into BeautifulSoup object
        return BeautifulSoup(response.text, 'html.parser')

    except requests.RequestException as e:
        # print error message if request fails
        if verbose:
            print(f"Error fetching {url}: {e}")
        return None


def extract_links(soup):
    """
    Extract links and names from a table on the webpage.

    Args:
        soup (BeautifulSoup): Parsed HTML content

    Returns:
        list: List of tuples containing (name, url) pairs. Names are transliterated
              from Hindi to English using unidecode.

    Example:
        >>> from bs4 import BeautifulSoup
        >>> html = '<table id="t1"><tr><td></td><td><a href="/test">
                    </a></td></tr></table>'
        >>> soup = BeautifulSoup(html, 'html.parser')
        >>> links = extract_links(soup)
        >>> len(links) > 0
        False
    """
    # return empty list if soup is None
    if soup is None:
        return []

    # find the table with id 't1'
    table = soup.find('table', id='t1')

    # return empty list if table not found
    if not table:
        return []

    # get all rows except first 4 header rows
    rows = table.find_all('tr')[4:]

    # initialize empty list for storing links
    link_list = []

    # loop over each row in the table
    for row in rows:
        # extract all cells from the row
        cells = row.find_all('td')

        # check if row has at least 2 cells
        if len(cells) >= 2:
            # find the link tag in the second cell
            link_tag = cells[1].find('a')

```

```
            # process link if found
            if link_tag:
                # extract and transliterate name from Hindi to English
                name = unidecode(link_tag.text.strip())

                # get href attribute
                href = link_tag['href']

                # construct full URL from base and relative URL
                full_url = urljoin(BASE_URL_PATH, href)

                # append tuple of name and URL to list
                link_list.append((name, full_url))

    # return list of extracted links
    return link_list


def scrape_table_data(soup, state, state_id, fy, district, block):
    """
    Scrape panchayat-level table data from a block webpage.

    Args:
        soup (BeautifulSoup): Parsed HTML content
        state (str): State name
        state_id (str): State code/ID
        fy (str): Financial year
        district (str): District name
        block (str): Block name

    Returns:
        list: List of dictionaries containing panchayat data with monthly
    figures

    Example:
        >>> soup = None
        >>> data = scrape_table_data(soup, "State", "01", "2023", "District", "
    Block")
        >>> len(data)
        0
    """
    # return empty list if soup is None
    if soup is None:
        return []

    # find the table with id 't1'
    table = soup.find('table', id='t1')

    # return empty list if table not found
    if not table:
        return []

    # get all rows except first 4 header rows
    rows = table.find_all('tr')[4:]

    # initialize empty list for storing data
    data_list = []

    # loop over each row in the table
    for row in rows:
        # extract all cells from the row
        cells = row.find_all('td')

```

```
        # check if row has at least 14 cells (name + 12 months)
        if len(cells) >= 14:
            # extract and transliterate panchayat name from Hindi to English
            panchayat = unidecode(cells[1].text.strip())

            # extract monthly data from columns 2-13 (12 months)
            monthly_data_list = [cell.text.strip() for cell in cells[2:14]]

            # create dictionary with all data for this panchayat
            data_list.append({
                'State': state,
                'State ID': state_id,
                'FY': fy,
                'District': district,
                'Block': block,
                'Panchayat': panchayat,
                'April': monthly_data_list[0],
                'May': monthly_data_list[1],
                'June': monthly_data_list[2],
                'July': monthly_data_list[3],
                'August': monthly_data_list[4],
                'September': monthly_data_list[5],
                'October': monthly_data_list[6],
                'November': monthly_data_list[7],
                'December': monthly_data_list[8],
                'January': monthly_data_list[9],
                'February': monthly_data_list[10],
                'March': monthly_data_list[11]
            })

    # return list of scraped panchayat data
    return data_list


def main(verbose=True):
    """
    Main function to orchestrate the hierarchical scraping of administrative
    data.

    Scrapes data at multiple levels: states -> districts -> blocks ->
    panchayats,
    and saves all collected data to a CSV file.

    Args:
        verbose (bool): If True, print status messages during scraping. Default
     is True.

    Returns:
        None. Saves output to CSV file.

    Example:
        >>> # This would run the full scraping process
        >>> # main(verbose=False)
    """
    # initialize empty list to store all scraped data
    all_data_list = []

    # loop over each state URL in the list
    for state_url in state_url_list:
        # parse URL to extract query parameters
        parsed_url = urlparse(state_url)

        # extract query parameters from URL
```

```
        query_params = parse_qs(parsed_url.query)

        # get state name from query parameters (default to 'Unknown')
        state = query_params.get('state_name', ['Unknown'])[0]

        # get state ID from query parameters (default to 'Unknown')
        state_id = query_params.get('state_code', ['Unknown'])[0]

        # get financial year from query parameters (default to 'Unknown')
        fy = query_params.get('fin_year', ['Unknown'])[0]

        # print status message for current state
        if verbose:
            print(f"Processing state: {state}")

        # fetch and parse state page
        state_soup = get_soup(state_url, verbose=verbose)

        # skip to next state if page fetch failed
        if not state_soup:
            if verbose:
                print(f"Failed to fetch state page for {state}. Skipping.")
            continue

        # extract district links from state page
        district_link_list = extract_links(state_soup)

        # print number of districts found
        if verbose:
            print(f"Found {len(district_link_list)} districts in {state}.")

        # loop over each district in the state
        for district_name, district_url in district_link_list:
            # print status message for current district
            if verbose:
                print(f"Processing district: {district_name} in {state}")

            # fetch and parse district page
            district_soup = get_soup(district_url, verbose=verbose)

            # skip to next district if page fetch failed
            if not district_soup:
                if verbose:
                    print(f"Skipping {district_name} due to fetch error.")
                continue

            # extract block links from district page
            block_link_list = extract_links(district_soup)

            # print number of blocks found
            if verbose:
                print(f"Found {len(block_link_list)} blocks in {district_name}.
")

            # loop over each block in the district
            for block_name, block_url in block_link_list:
                # print status message for current block
                if verbose:
                    print(f"Processing block: {block_name} in {district_name},
{state}")

                # fetch and parse block page
                block_soup = get_soup(block_url, verbose=verbose)
```

```
289
290                 # skip to next block if page fetch failed
291                 if not block_soup:
292                     if verbose:
293                         print(f"Skipping {block_name} due to fetch error.")
294                     continue
295
296                 # scrape panchayat data from block page
297                 panchayat_data_list = scrape_table_data(
298                     block_soup, state, state_id, fy, district_name, block_name
299                 )
300
301                 # add panchayat data to overall data list
302                 all_data_list.extend(panchayat_data_list)
303
304                 # print number of panchayats collected
305                 if verbose:
306                     print(f"Collected data for {len(panchayat_data_list)}
    panchayats in {block_name}.")
307
308     # check if any data was collected
309     if all_data_list:
310         # convert list of dictionaries to DataFrame
311         df_panchayat = pd.DataFrame(all_data_list)
312
313         # define output file path
314         output_path = "put/your/output/path/here.csv"
315
316         # save DataFrame to CSV file
317         df_panchayat.to_csv(output_path, index=False, encoding='utf-8')
318
319         # print success message with row count
320         if verbose:
321             print(f"Data saved to {output_path} with {len(df_panchayat)} rows."
    )
322     else:
323         # print message if no data was collected
324         if verbose:
325             print("No data collected.")
326
327
328 if __name__ == "__main__":
329     # run main function when script is executed
330     main()
```