snotel

April 3, 2024

The heading should be realted to “workflow”

1 Analyzing Snowpack Data with Python

1.1 Chapter 1 : Introduction

This chapter introduces the significance of analyzing snowpack data, focusing on the Snow Telemetry (SNOTEL) and California Data Exchange Center (CDEC) stations. These stations play a crucial role in monitoring snowpack dynamics across the Western United States, providing vital data for water resource management, climatological research, and environmental conservation.

1.1.1 1.2. Objectives

The main objective should be also to explain how Snowpack plays an important role in SWE Workflow

- Understand the importance of snowpack data.
- Learn how to automate the retrieval, processing, and analysis of snowpack data using Python.

1.2 Chapter 2: Setting Up Your Environment

1.3 2.1.1. Getting Started

1.3.1 2.1. Prerequisites

. Python 3.x

. Libraries: math, json, requests, pandas, csv, io, os, dask

. A working directory for storing downloaded files

1.3.2 2.2. Environment Preparation

Instructions on setting up the Python environment, installing necessary libraries using pip, and organizing your workspace for efficient data handling.

1.3.3 2.3.1. Setup

[3]:
```
#Step 0: Import Libraries
import requests
import pandas as pd
import json
```

[4]:
```
# Constants
WORK_DIR = 'C:\\Users\\Lenovo\\Documents\\SNOTEL Training'
```

```
STATION_JSON_URL = "https://wcc.sc.egov.usda.gov/awdbRestApi/services/v1/
 ↪stations?
 ↪activeOnly=true&returnForecastPointMetadata=false&returnReservoirMetadata=false&returnStati
OUTPUT_JSON_FILE = f'{WORK_DIR}/all_snotel_cdec_stations.json'
OUTPUT_CSV_FILE = f'{WORK_DIR}/all_snotel_cdec_stations.csv'
```

Please add what does Work_dir means here

1.4 Chapter 3: Fetching Station Data

1.4.1 3.1. Understanding APIs

A brief introduction to APIs (Application Programming Interfaces), focusing on how the SNOTEL and CDEC API provides access to station data.

We dont have to explain the fundamentals of HTTP and fetch

1.4.2 3.2. Writing the Download Function

Explains the `download_station_json` function, which uses the `requests` library to fetch JSON data from the SNOTEL and CDEC API. This section includes handling HTTP responses to ensure successful data retrieval.

[5]:
```
# Step 1: Download Station Data
def download_station_json():
    response = requests.get(STATION_JSON_URL)
    if response.status_code == 200:
        with open(OUTPUT_JSON_FILE, 'w') as json_file:
            json.dump(response.json(), json_file, indent=2)
        print("Data downloaded and saved.")
    else:
        print("Failed to download data. Status code:", response.status_code)
```

1.5 Chapter 4: Transforming Data Formats

1.5.1 4.1. From JSON to CSV

Discusses the rationale behind converting JSON data to CSV format for easier manipulation and analysis. It includes a step-by-step guide on using the `pandas` library for the conversion process.

1.5.2 4.2. Implementing the Conversion

Detailed explanation of the `convert_json_to_csv` function, illustrating how to read JSON from a file, normalize nested JSON into a flat table, and save the result as a CSV file.

[6]:
```
# Step 2: Convert JSON to CSV
def convert_json_to_csv():
    with open(OUTPUT_JSON_FILE, 'r') as json_file:
        data = json.load(json_file)
    df = pd.json_normalize(data)
    df.to_csv(OUTPUT_CSV_FILE, index=False)
    print("JSON converted to CSV.")
```

A map showing all the relevent stations pointed out
greatly helps the reader to understand where the stations are located.

1.6 Chapter 5: Filtering Relevant Stations

1.6.1 5.1. The Need for Filtering

Outlines why filtering stations based on specific criteria (e.g., geographic coordinates) is necessary for targeted analysis.

1.6.2 5.2. Crafting the Filter

Provides an in-depth look at the `filter_stations` function, demonstrating how to use `pandas` to apply filters to the CSV data. This section explains selecting rows based on conditions, such as latitude greater than 40 degrees.

[7]:
```
# Step 3: Filter Stations (Example filtering logic; adjust as needed)
def filter_stations():
    df = pd.read_csv(OUTPUT_CSV_FILE)
    # Example filter: Latitude > 40
    filtered_df = df[df['latitude'] > 40]
    print(filtered_df.head())
```

1.7 Chapter 6: Conclusion and Next Steps

1.7.1 6.1. Review

Summarizes the script's functionality and the theoretical knowledge provided in the previous chapters, reinforcing the importance of each step in the data analysis process.

[8]:
```
# Step 4: Main function to execute the steps
def main():
    download_station_json()
    convert_json_to_csv()
    filter_stations()

if __name__ == "__main__":
    main()
```

What does filter_stations mean here?

```
Data downloaded and saved.
JSON converted to CSV.
    stationTriplet stationId stateCode networkCode                    name  \
2     0010:ID:COOP      0010        ID        COOP  Aberdeen Experimnt Stn
3    1F01A:BC:SNOW     1F01A        BC        SNOW            Aberdeen Lake
6    13E19:ID:SNOW     13E19        ID        SNOW            Above Gilmore
7    15B07:ID:SNOW     15B07        ID        SNOW             Above Roland
9  06110500:MT:BOR  06110500        MT         BOR              Ackley Lake

  dcoCode    countyName           huc  elevation  latitude  longitude  \
2      ID       Bingham  1.704021e+11     4410.0  42.95000 -112.83333
3      OR       UNKNOWN           NaN     4298.0  50.14733 -119.05340
6      ID         Lemhi  1.706020e+11     8289.0  44.45615 -113.30097
7      ID      Shoshone  1.701030e+11     4347.0  47.38507 -115.66405
```

```
9      MT  Judith Basin  1.004010e+11     4300.0  46.95741 -109.94062

   dataTimeZone pedonCode shefId              beginDate                endDate
2           NaN       NaN  ABDI1  1914-01-01 00:00:00.0  2100-01-01 00:00:00.0
3           NaN       NaN  ABLQ2  1939-04-01 00:00:00.0  2100-01-01 00:00:00.0
6           NaN       NaN  ABGI1  1961-01-01 00:00:00.0  2100-01-01 00:00:00.0
7           NaN       NaN  ABRI1  1926-03-01 00:00:00.0  2100-01-01 00:00:00.0
9           NaN       NaN    NaN  1938-06-01 00:00:00.0  2100-01-01 00:00:00.0
```

This code snippet is a basic Python script that outlines a process for handling and transforming weather station data. It consists of a main function, `main()`, that orchestrates three key steps in the data handling process: downloading JSON data, converting that JSON data into a CSV format, and finally filtering the stations based on certain criteria. The script is designed to be executed as a standalone program, as indicated by the `if __name__ == "__main__":` condition, which ensures that the `main()` function is called only when the script is run directly, and not when imported as a module.

This paragraph is too generic, it should be followed up by relating it to how it integrates with the workflow

The snippet also includes an example output that showcases the result of the operations, particularly the conversion of JSON to CSV format. This output displays a table with selected columns such as `stationTriplet`, `stationId`, `stateCode`, `networkCode`, `name`, `dcoCode`, `countyName`, `huc`, `elevation`, `latitude`, `longitude`, `dataTimeZone`, `pedonCode`, `shefId`, `beginDate`, and `endDate`. These columns represent various attributes of weather stations, including their IDs, names, geographic locations (state, county, latitude, longitude), elevation, and the time range for which data is available.

The script presumably performs the following operations: 1. **Downloading JSON Data**: It fetches weather station data in JSON format from a remote source or local storage. 2. **Converting JSON to CSV**: The JSON data is then converted into a more universally readable comma-separated values (CSV) format, facilitating easier data manipulation and analysis. 3. **Filtering Stations**: Lastly, the script filters the stations based on predefined criteria, which could include factors like geographic location, data availability period, or specific attributes of the stations.

This process is essential for data preparation, especially in meteorological or geographical information system (GIS) applications, where handling large datasets and converting them into a more accessible format is a preliminary step for further analysis or visualization.

This structured approach aims to equip readers with both the theoretical background and practical skills needed to harness Python for snowpack data analysis, promoting a deeper understanding of environmental data's role in research and decision-making.

The conclusion part should relate to how its being used in the workflow