**fire\_index\_mapping.py**

Created comprehensive code that combines the provided code and functions as follows:

* uses the code in the attached open-meteo Jupyter Notebook to get the current weather parameters needed by the code in following attached Jupyter Notebooks:
  + the NRDS4\_FDI Jupyter Notebook for getting current values of FFDI
  + the Canadian Forest Fire Weather Index (FWI) System Jupyter Notebook for getting current values of FWI
  + the mFFWI Jupyter Notebook for getting the current values of mFFWI
* and then uses Leaflet and Javascript to create an interactive color map of the State of California with selectable overlays of the current FFDI, FWI, and nFFWI values.

**How to Use:**

1. **get\_california\_geojson\_from\_api() Function:**

* This function takes the API URL as an argument (defaulting to the raw.githubusercontent.com URL you provided).
* It uses requests.get() to fetch the data from the URL.
* response.raise\_for\_status() checks if the request was successful (status code 200 OK). If not, it raises an exception.
* response.json() parses the JSON response into a Python dictionary.
* It includes error handling (using try...except) to catch network or API errors and prints an error message.

1. **Install Shapely:** Install the shapely library:

pip install shapely

1. **Combined Code:** Code **Part 1**, **Part 2**, and **Part 3** are contained in a single Python file (fire\_index\_mapping.py).
2. **Install Libraries:**

pip install openmeteo-requests requests-cache pandas retry-requests numpy folium

1. **Run the Script: Execute the Python file from terminal:**

python fire\_index\_mapping.py

This will generate three HTML files: ffdi\_map.html, fwi\_map.html, and mffwi\_map.html. These files can be opened in a web browser to view the interactive maps.

**Important Notes:**

1. **Import shapely:** Added imports for the shapely library to work with geometric objects:

import shapely.geometry

import shapely.ops

1. **create\_fire\_index\_map() Function:**
   * **GeoJSON Argument:** Now takes a california\_geojson argument, which is the loaded GeoJSON data for California.
   * **Shapely Polygon:** Creates a shapely.geometry.Polygon object from the GeoJSON data using shapely.geometry.shape(). This polygon will be used to check if grid points are inside California.
   * **Point-in-Polygon Check:** Inside the loop, it creates a shapely.geometry.Point object for each grid point's (lon, lat) coordinates. Then, it uses the california\_polygon.contains(point) method to check if the point lies within the California polygon.
   * **Conditional Marker:** The folium.CircleMarker is only added if the contains() method returns True (i.e., the point is inside California).
   * **GeoJSON Boundary:** The folium.GeoJson layer now uses the california\_geojson data to display the more accurate California boundary. The style\_function is used to make the boundary a simple gray line without fill.

* **get\_weather\_data() Function:**
  + Takes hourly parameters in params to fetch hourly temperature, relative humidity, precipitation, and wind speed data.
  + The function also includes daily parameters to fetch daily maximum and minimum values for temperature, relative humidity, precipitation, and wind speed.
  + It returns current\_data (current conditions) and daily\_data (including Tmax, RHmin, etc.).
* **Main Execution Block (if \_\_name\_\_ == "\_\_main\_\_":)**
  + The california\_geojson variable now gets its value from the get\_california\_geojson\_from\_api() function.
  + There's a check: if california\_geojson is None:, which handles the case where the API request failed. If the data couldn't be fetched, the script prints an error message and exits.
  + Load GeoJSON: The code now loads the GeoJSON data from a file named california.geojson using json.load(). Make sure you have this file in the same directory as your Python script.
  + Pass GeoJSON to create\_fire\_index\_map(): The loaded california\_geojson is passed to the create\_fire\_index\_map() function when creating each map.
  + Calls get\_weather\_data() to get both current and daily data.
  + Uses tmax (daily maximum temperature) and rhmin (daily minimum relative humidity) from daily\_weather\_data in calculate\_fire\_danger\_nfdrs4().
  + Continues to use the current day's wind speed and precipitation for FWI calculations (you might adjust this if you have more specific requirements).
  + The calc\_ffwi function (for mFFWI) is also updated to use Tmax and RHmin in Fahrenheit.
* **Error Handling:** The code includes basic error handling to catch issues when fetching or processing data for specific grid points.
* **Grid Resolution:** The grid resolution (0.5 degrees in this example) is a balance. You can increase the resolution for more detail but it will increase processing time and map file size.

Here is a california\_grid definition that uses a **0.25-degree spacing** instead of 0.5 degrees. This will create a denser grid, resulting in a more detailed map, but it will also significantly increase processing time and the size of the generated HTML files.

# Spacing: Approximately 0.25 degrees for more detail

california\_grid = [(lat, lon) for lat in np.arange(32.5, 42.1, 0.25) for lon in np.arange(-124.5, -114, 0.25)]

* np.arange(32.5, 42.1, 0.25): This creates a sequence of latitude values starting from 32.5, going up to (but not including) 42.1, with a step of 0.25 degrees.
* np.arange(-124.5, -114, 0.25): This does the same for longitude values, starting from -124.5, going up to (but not including) -114, with a step of 0.25 degrees.
* [(lat, lon) for lat in ... for lon in ...]: This is a list comprehension that creates all the (latitude, longitude) pairs by combining each latitude with each longitude value.

**How Many Points?**

Calculation of how many grid points this will generate:

* Latitude range: 42.1 - 32.5 = 9.6 degrees
* Longitude range: -114 - (-124.5) = 10.5 degrees
* Number of latitude steps: 9.6 / 0.25 = 38.4 (approximately 39 steps)
* Number of longitude steps: 10.5 / 0.25 = 42 steps
* Total grid points: 39 \* 42 = **1638**

So, this 0.25-degree grid will have approximately **1638** grid points.

**Considerations:**

* **Processing Time:** Fetching and processing data for 1638 points will take considerably longer than the previous example. Be prepared to wait, especially since you're making API calls to Open-Meteo.
* **File Size:** The resulting HTML files will be significantly larger, potentially impacting loading times in your browser.
* **API Limits:** Again, be mindful of Open-Meteo's API usage limits if you're running this frequently or with an even denser grid.
* **Memory:** Processing and storing data for many grid points can consume more memory. Ensure your system has enough RAM, especially if you increase the density further.

**Even Denser Grid (0.125 degrees - Example):**

If you need even finer detail, you could use a 0.125-degree spacing, but be aware of the increasing computational cost:

california\_grid = [(lat, lon) for lat in np.arange(32.5, 42.1, 0.125) for lon in np.arange(-124.5, -114, 0.125)]

This would result in approximately **6552** grid points (double the steps in each direction, hence four times the number of points).

**Recommendation:**

Remember to clear the cache if testing with different grid resolutions. You can clear the cache using:

rm .cache

* **Colormap:** You can customize the colormap in the create\_fire\_index\_map function to better represent the fire index values.
* **Simplifications:** For brevity, the code makes some simplifications (e.g., assuming the current temperature and relative humidity are close to the daily maximum and minimum, respectively). You might refine these based on your specific needs and data availability.
* **API Usage:** Be mindful of the Open-Meteo API usage limits if you are making a large number of requests. The caching mechanism helps reduce the number of requests.
* **Boundary:** The California boundary is an approximation. If you need a very precise boundary, use a more detailed GeoJSON file for California.