Table of contents

Author: Guillermo Romero For help or guidance I can be reached at: Linkedin, Email

Huggingface Web App:

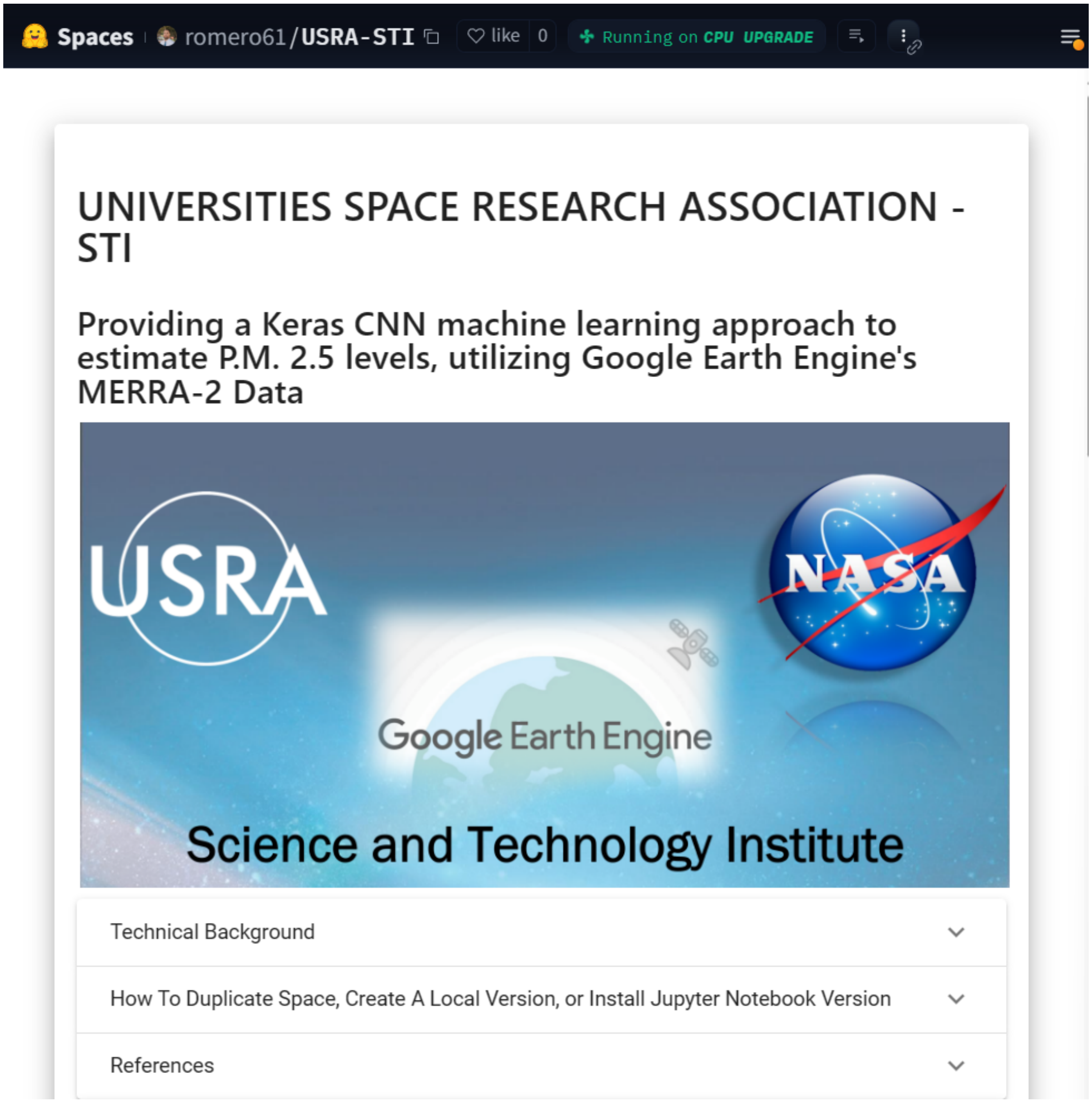
https://huggingface.co/spaces/romero61/USRA-STI

Github Repository:

https://github.com/romero61/USRA-STI

References:

- "GESDISC." Nasa.gov, 2023
- Global Modeling and Assimilation Office (GMAO) (2015), *inst3_3d_asm_Cp: MERRA-2 3D IAU State, Meteorology Instantaneous 3-hourly (p-coord, 0.625x0.5L42), version 5.12.4*, Greenbelt, MD, USA: Goddard Space Flight Center Distributed Active Archive Center (GSFC DAAC), doi: 10.5067/VJAFPLI1CSIV.
- Wu, Q., (2020). geemap: A Python package for interactive mapping with Google Earth Engine. The Journal of Open Source Software, 5(51), 2305. https://doi.org/10.21105/joss.02305
- Wu, Q., Lane, C. R., Li, X., Zhao, K., Zhou, Y., Clinton, N., DeVries, B., Golden, H. E., & Lang, M. W. (2019). Integrating LiDAR data and multi-temporal aerial imagery to map wetland inundation dynamics using Google Earth Engine. Remote Sensing of Environment, 228, 1-13. https://doi.org/10.1016/j.rse.2019.04.015 (pdf | source code)



UNIVERSITIES SPACE RESEARCH ASSOCIATION - STI

Providing a Keras CNN machine learning approach to estimate P.M. 2.5 levels, utilizing Google Earth Engine's MERRA-2 Data

Technical Background

How To Duplicate Space, Create A Local Version, or Install Jupyter Notebook Version

References

Key Areas:

In public/01-main.py:

```
@solara.component
def CombinedThread(running,...
final_df = CNN(model_status,...))
```

The function CNN() comes from the file public/ee_cnn.py file. The function uses other functions within the same file to call the API and preprocess the data.

Scenario 1:

If the preprocessing remains exactly the same, and the CNN itself needs to be updated then this can be updated in `pages/ee_cnn.py` beginning at line :

```
345path = os.path.join(SCALAR,"global_scalars2.csv")
```

Note: V3, V4, V5 have the added KDTree solution for mapping lat/lon scalar points to points available from Earth Engine data.

Scenario 2:

If the collection and/or bands that are used need to be updated, the functions currently used and that would need to be updated are `fetch_slv_data_hourly()` and `fetch_aer_data_hourly()`.

Scenario 3:

If instead of pulling by hour you would like to pull the data all at once and process, the `fetch_slv_data` and `fetch_aer_data` can be substituted. There is code in place in the CNN function that is commented out that can be used to then process by hour.

Scenario 4:

If the preprocessing of Earth Engine data changes and/or bands change, then the primary function to be updated is `process_data()`. After preprocessing the Earth Engine data, to work with with the CNN the data is then transformed to a numpy array so this will require update as well in the CNN() function starting at line **297**.

01-main.py:

Overview:

The 01-main.py script is part of a web-based application designed for geospatial data analysis and machine learning, focusing on environmental data. It integrates various libraries to handle data processing, machine learning, and interactive web elements. The application likely runs in a Jupyter or similar interactive Python environment.

Key Components

1. Imports

Ensure you have all the required libraries installed:

Geospatial libraries: `ee, geemap, geopandas, shapely, cartopy`

Data handling: `pandas, numpy, h5py`

Machine Learning: `keras, tensorflow`

Web application and interactivity: `ipywidgets, ipyleaflet, solara`

Others: `rasterio, rioxarray, psutil, json, geojson, datetime, threading, multiprocessing`

2. Memory Usage Monitoring log_memory_usage():

A function to log the memory usage of the process. Useful for debugging and optimizing performance.

3. Google Earth Engine Initialization

The script initializes Google Earth Engine (GEE) using credentials from an environment variable (EEPRIVATE_KEY). Ensure this variable is set in your environment or modify the script to suit your authentication method.

4. Project Structure

The script sets up paths related to the project and adds a module path to the Python path. Ensure the directory structure of your project matches what the script expects or modify accordingly.

5. User Interface Components with Solara

The script uses solara to define reactive variables and components for the user interface. Familiarize yourself with solara for UI element customization.

6. Model Control and Status Bar

Components for controlling the model's execution (ModelControl) and displaying its status (StatusBar). Understand the flow and signals between these components for effective debugging and enhancements.

7. Threading for Model Execution

CombinedThread: Manages threading for the model's execution to keep the UI responsive. Review Python's threading and event handling if you need to modify this behavior.

8. Main Page Layout

Page: The main component that lays out the entire UI, including instructions, image banners, expansion panels, date selection UI, map UI, and status displays. This is where you'll make changes to the overall layout and flow of the application.

9. Visualization and Map Components

Components at the end of the script are responsible for displaying results, including interactive charts and maps. If you need to enhance the visualization aspect, focus on these components.

ee_cnn.py

Overview

The `ee_cnn.py` script is a core component of a geospatial data analysis application, focusing on environmental data processing and prediction using Convolutional Neural Networks (CNNs) with data from Google Earth Engine. It's designed to fetch, preprocess, and use machine learning models on environmental data for predictive analysis.

Key Components

1. Imports

The script begins by importing necessary Python libraries for data handling, machine learning, multiprocessing, and more. Ensure these libraries are installed and properly configured in your environment:

Geospatial libraries: `ee, geemap, geopandas, shapely, rasterio, rioxarray`

Data handling: `pandas, numpy, h5py`

Machine Learning: `keras, tensorflow`

Others: `datetime, warnings, xarray, psutil`

2. Memory Usage Function

`log_memory_usage():` A function to monitor and log the memory usage of the process, aiding in performance optimization and debugging.

3. Data Fetching Functions

Several functions are defined to fetch different types of data from the Google Earth Engine:

`fetch_slv_data():` Fetches single-level variables (SLV) data.

`fetch_aer_data():` Fetches aerosol data.

`fetch_slv_data_hourly():` Fetches hourly SLV data.

`fetch_aer_data_hourly():` Fetches hourly aerosol data.

4. Data Processing

`process_data():` Processes the fetched data by combining, masking, and transforming it to prepare for the CNN model. It includes steps like normalization, adding wind calculations, and combining latitude and longitude information.

5. Solar Zenith Angle Calculation

`get_SZA():` Calculates the Solar Zenith Angle, which is a measure of the amount of sunlight reaching the Earth's surface at a specific location and time.

6. Normalization Function

`normalize()`: Normalizes the data frame values between 0 and 1 based on the maximum and minimum values provided.

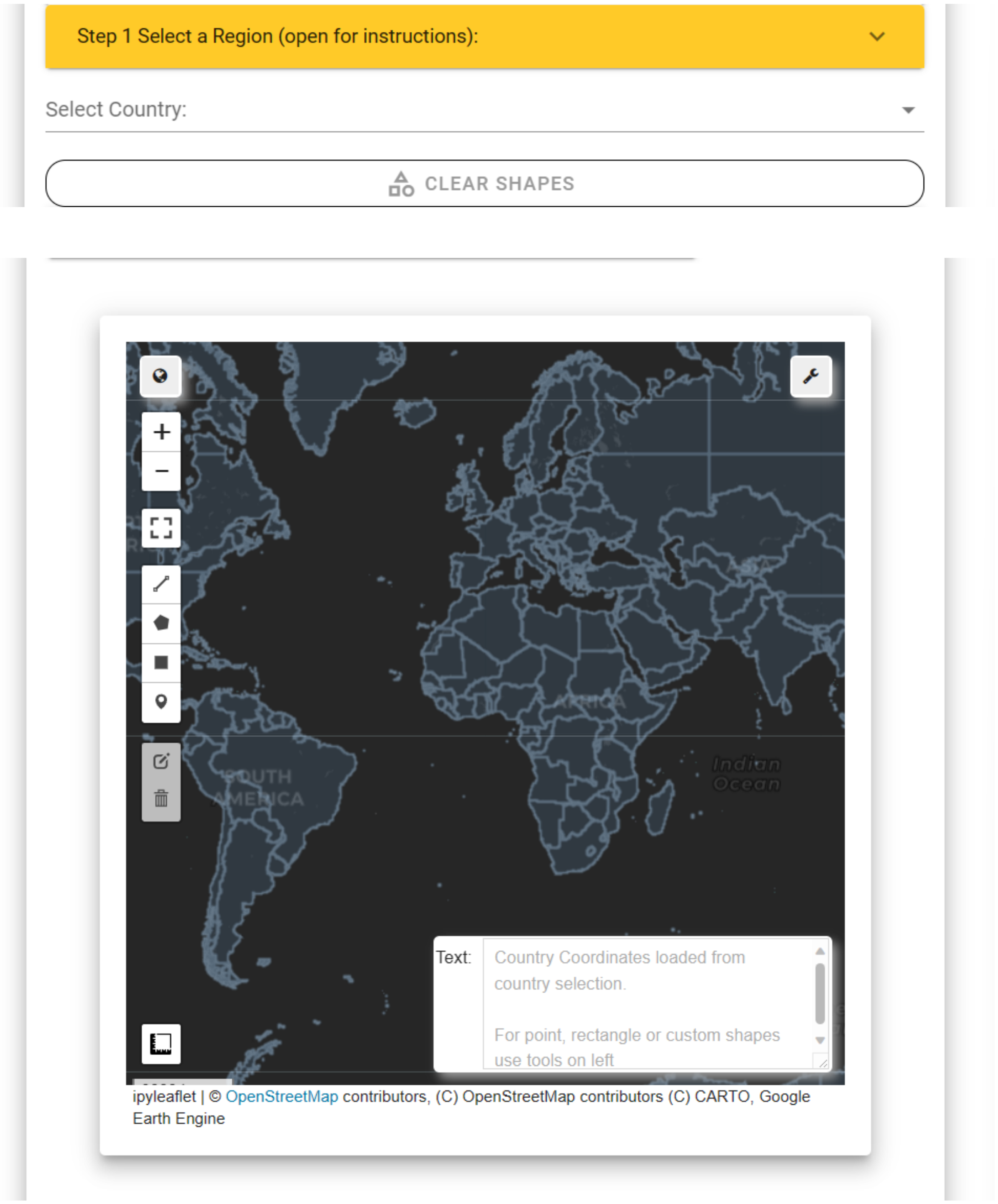
7. Custom Loss Function for Keras

`customLoss1()`: Defines a custom loss function for the Keras model, which is used during the training or prediction phase.

8. CNN Model Function

`CNN()`: The main function that orchestrates the data fetching, processing, and prediction using various CNN models. It handles different versions of the model, manages multiprocessing for efficiency, and compiles results into a final data frame.

map_ui.py



Step 1 Select a Region (open for instructions):

Select Country:

CLEAR SHAPES

Text: Country Coordinates loaded from country selection.

For point, rectangle or custom shapes use tools on left

ipyleaflet | © OpenStreetMap contributors, (C) OpenStreetMap contributors (C) CARTO, Google Earth Engine

Overview

The map_ui.py script is a component of a geospatial data analysis application, focusing on providing an interactive map interface for users to select regions, countries, or draw custom shapes. It utilizes the geemap and ipyleaflet libraries for map rendering and interaction, integrated into a web application environment.

The map_ui.py script provides a sophisticated interactive map interface for the application, allowing users to select specific regions or countries for further analysis. It's a crucial component for any geospatial analysis application, providing the means for users to specify the area of interest. As you work with or extend this script, keep user interaction and experience in mind, ensuring that the map remains intuitive, responsive, and informative.

Key Components

1. Imports

The script imports necessary libraries for map interaction and widget controls:

- `ee:` Google Earth Engine library for accessing geospatial data.
- `geemap:` A Python package for interactive mapping with Google Earth Engine.
- `ipywidgets and ipyleaflet`: For creating interactive UI elements and map controls.
- `solara:` Likely used for reactive programming patterns in the application.

2. MapUI Class

The MapUI class is the core of the script, encapsulating all the functionality related to the map interface.

Initialization (`__init__`):

- Initializes the map using geemap.Map().
- Sets up the selected coordinates and country feature based on user interaction.
- Calls _initialize_ui() to set up the initial UI components.

UI Initialization (`_initialize_ui`):

- Sets up the map style, adds a base map, and overlays a layer of countries.
- Creates a textarea widget to display selected coordinates or country information.
- Adds a selection dropdown for countries and a button to clear drawn shapes.
- Attaches event handlers for drawing on the map and selecting countries.

Drawing Handler (`handle_draw`):

- Handles the 'drawn' event on the map, updating the selected coordinates or country based on the drawn shape or point.
- Updates the textarea widget with the new selection information.

Clear Shapes Handler (`on_clear_click`):

- Clears all drawn shapes from the map and resets the UI elements to their default state.

Country Selection Handler (`on_country_change`):

- Updates the map and UI based on the selected country from the dropdown.
- Centers the map on the selected country and updates the textarea widget with the country's name and coordinates.

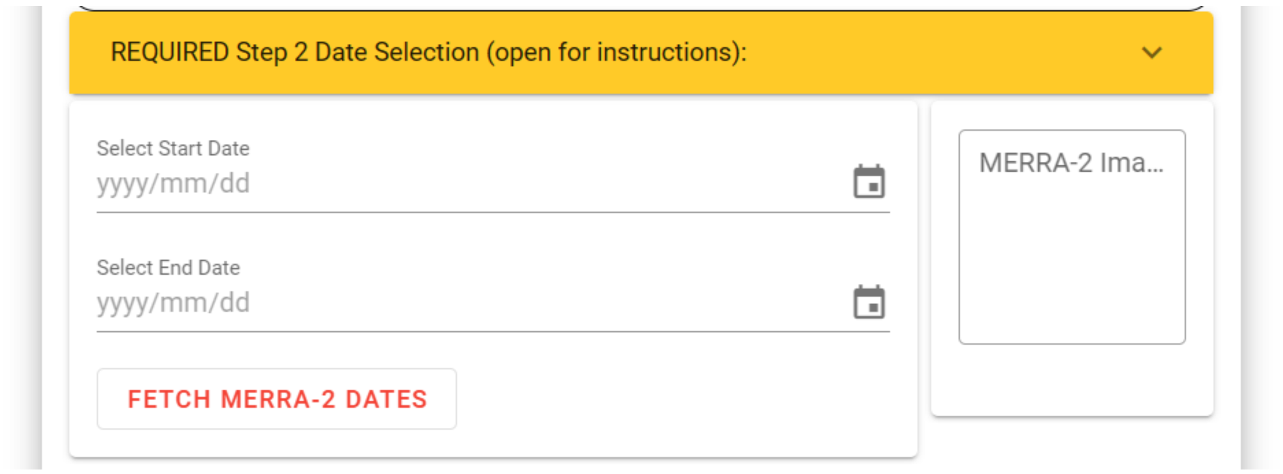
Display Map (`display_map`):

- Returns the configured map for rendering in the application.

Development Tips

- Map Customization: Familiarize yourself with the geemap and ipyleaflet libraries for any further customization or additional features you might want to add to the map.
- Event Handling: Understand the event handling mechanism for drawing and selection on the map. This is crucial for extending or modifying the interaction behavior.
- Reactive State Management: Notice the use of reactive state variables (selected_coordinates, selected_country_feature). Ensure that any new interactive features you add follow this reactive pattern for a consistent and responsive UI.
- Testing: Test any changes in an environment where you can interact with the map, such as Jupyter Notebook or a web application frontend. Ensure that the map loads correctly and that all interactive elements respond as expected.

merra_date_ui.py



Overview

The `merra_date_ui.py` script is part of a geospatial data analysis application, focusing on providing a user interface for selecting date ranges to fetch MERRA (Modern-Era Retrospective analysis for Research and Applications) data. It utilizes the `solara` and `reacton.ipyvuetify` libraries for creating reactive and interactive UI components.

The **`merra_date_ui.py`** script is a crucial component for any application requiring users to interactively select date ranges for fetching geospatial data, specifically MERRA data in this case. It provides a user-friendly interface for date selection, leveraging modern UI libraries and reactive programming to create a smooth user experience. As you work with or extend this script, keep in mind the importance of user feedback, error handling, and the overall integration with the data fetching and processing parts of your application.

Key Components

1. Imports

The script imports necessary libraries for creating the UI and interacting with Google Earth Engine:

- *solara:* Used for reactive programming patterns in the application.
- *reacton.ipyvuetify*: A library for creating Vuetify-based components in Jupyter and other Python environments.
- *ee*: Google Earth Engine library for accessing geospatial data.

2. MerraDateUI Class

The MerraDateUI class encapsulates all functionality related to the date selection interface.

Initialization (`__init__`):

- Initializes reactive variables for managing the state of calendar widgets and selected dates.
- Holds a reference to the available_dates reactive variable, which will store the fetched date ranges.

Date Input Component (`Date_Input`):

- Creates a text field with a calendar icon for users to input or select dates.
- Opens the calendar widget when the text field or calendar icon is clicked.

Calendar Component (`Calendar`):

- Provides a calendar widget allowing users to pick a date.
- Updates the selected start or end date based on user interaction.

Fetch Button Component (`FetchButton`):

- A button that, when clicked, fetches the available MERRA dates based on the selected start and end dates.
- Updates the `available_dates` reactive variable with the fetched dates.

Dates Output Component (`DatesOutput`):

- Displays the available MERRA dates fetched by the user.
- Uses a textarea component to list the dates.

Main UI Component (`MerraDateUIComponent`):

- Orchestrates the layout and rendering of the date input fields, calendar widgets, fetch button, and dates output.
- Utilizes `solara.Columns` and `solara.Card` for layout.

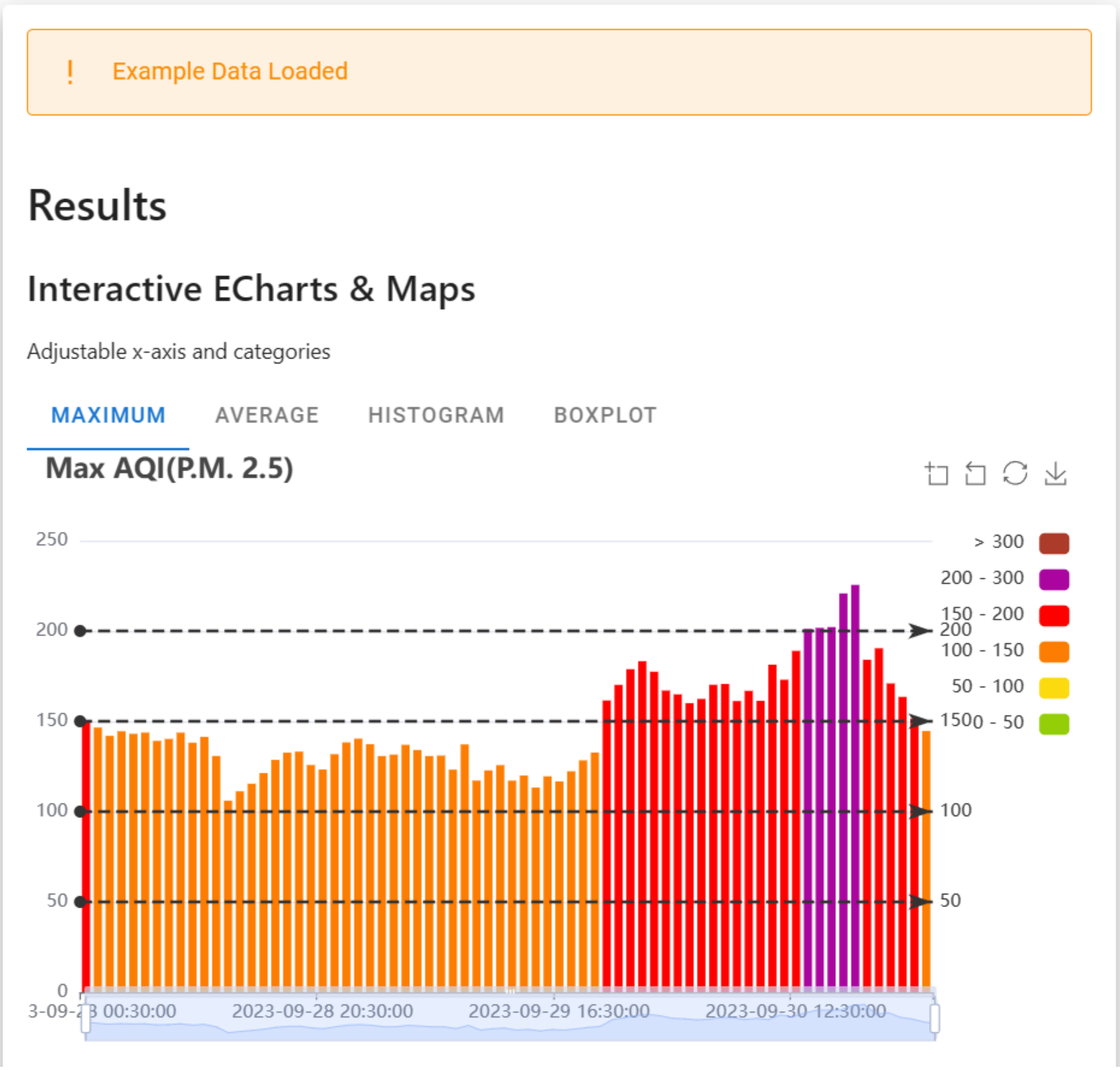
Display UI Function (`display_ui`):

- Returns the main UI component for rendering in the application.

Development Tips

- Reactive Programming: Understand the reactive programming paradigm used by solara. Reactivity is key to making the UI dynamic and responsive to user interactions.
- Google Earth Engine Integration: Familiarize yourself with the MERRA datasets available in Google Earth Engine to understand what data the script is fetching and how it might be used in the application.
- UI Customization: If you need to customize the UI or add new components, ensure they are integrated into the reactive framework provided by solara and reacton.ipyvuetify.
- Error Handling: Consider adding error handling for cases where data might not be available for the selected date range or other unexpected issues.

data_vizui.py



! Example Data Loaded

Results

Interactive ECharts & Maps

Adjustable x-axis and categories

Overview

The **`data_vizui.py`** script is a component of a geospatial data analysis application, focusing on visualizing environmental data, particularly PM2.5 air quality index data. It utilizes various libraries to create interactive charts and graphs, providing insights into the data's distribution, trends, and patterns.

The **`data_vizui.py`** script is a sophisticated component for visualizing environmental data, particularly focusing on air quality indices like PM2.5. It provides interactive and detailed insights into the data, helping users understand patterns and distributions over time and across different conditions. As you work with or extend this script, maintain a focus on data integrity, user interaction, and the overall narrative that the visualizations convey about the data.

Key Components

1. **Imports**

The script imports necessary libraries for data handling and visualization:

- **`numpy`** and **`pandas`**: For data manipulation and analysis.
- **`matplotlib.pyplot`**: For creating static plots (not directly used in the class but might be useful for extending the script).
- **`geemap`** and **`ee`**: For integrating with Google Earth Engine.
- **`ipywidgets`**: For creating interactive UI elements.
- **`solara`**: Likely used for reactive programming patterns in the application.
- **`datasets`**: For loading datasets, possibly used for demonstration or testing.

2. **EarthEngineDataVizUI Class**

The **`EarthEngineDataVizUI`** class encapsulates all functionality related to data visualization.

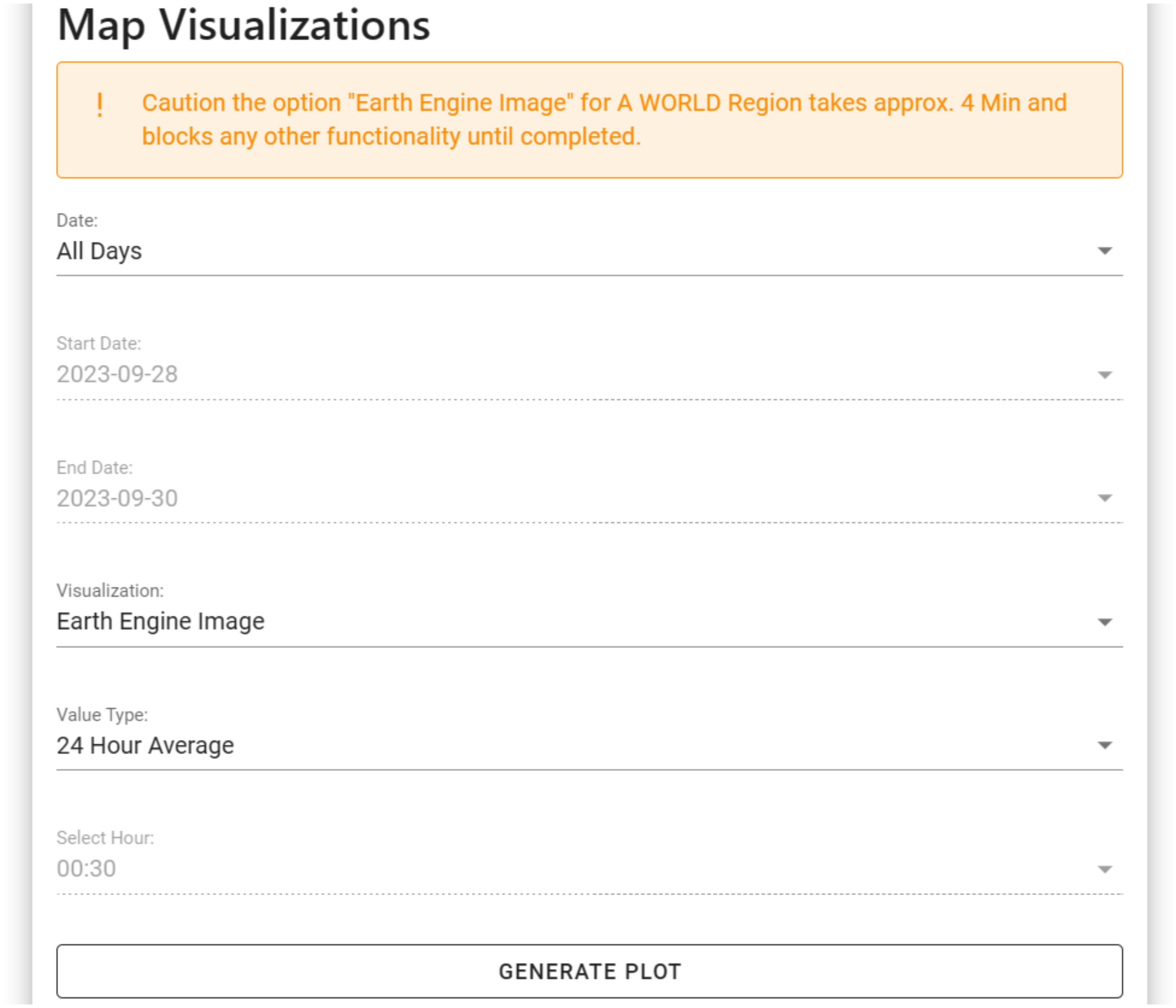
- **Initialization (`__init__`)**:
  - Initializes the class with a dataframe and a flag indicating if the data is available.
  - If data is available, it processes the dataframe to extract or format necessary columns.
  - If no data is available, it attempts to load a sample dataset for visualization.
- **Chart Options Generators (`generate_chart_options`, `generate_boxplot_options`)**:
  - These functions create configuration objects for different types of charts, including bar charts, histograms, and boxplots.
  - The configurations are tailored for use with a charting library that understands these options (likely ECharts or a similar JavaScript-based visualization library).
- **Data Preparation (`prepare_boxplot_data`)**:
  - Prepares data specifically for boxplot visualization, calculating statistical values like quartiles and extremes.

- **Data Processing and Chart Generation (`process_data_and_generate_charts`)**:
  - Main function that processes the dataframe to extract daily and hourly statistics.
  - Generates various charts based on the processed data, including maximum and average AQI, histograms, and boxplots.
  - Utilizes the `solara.lab.Tabs` component to organize different visualizations into tabs for a cleaner interface.

Development Tips

- **Understanding Visualization Libraries**: Familiarize yourself with the charting library used (likely ECharts or a similar library) to understand how the chart options are structured and how to modify or extend them.
- **Data Integrity**: Ensure that the data passed to the class is clean and well-formatted. The class expects a specific structure, particularly when setting indices and extracting date and time information.
- **Reactive Integration**: The class is likely used in a reactive environment. Understand how changes in the data or user interaction might trigger re-rendering or updating of the visualizations.
- **Extending Visualizations**: If you need to add new types of visualizations or customize existing ones, focus on the chart options generators and ensure that any new data processing aligns with the expected format of the visualization library.

map_visuals.py


Map Visualizations

! Caution the option "Earth Engine Image" for A WORLD Region takes approx. 4 Min and blocks any other functionality until completed.

Date:
All Days

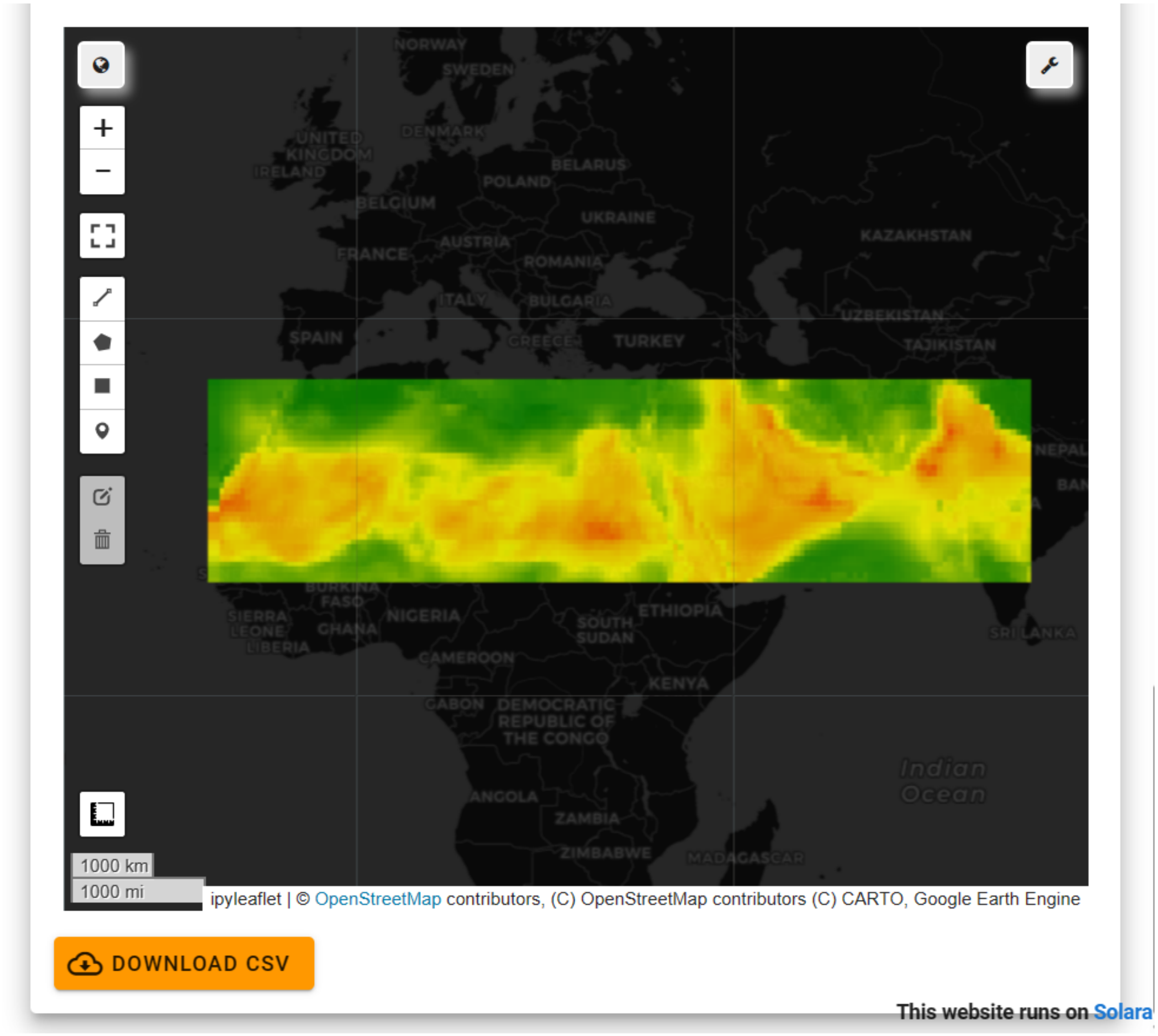
Start Date:
2023-09-28

End Date:
2023-09-30

Visualization:
Earth Engine Image

Value Type:
24 Hour Average

Select Hour:
00:30



Overview

The **map_visuals.py** script is part of a geospatial data analysis application, focusing on visualizing environmental data on a map interface. It utilizes various libraries to create interactive maps and plots, providing insights into the data's spatial distribution and trends.

The **map_visuals.py** script is a crucial component for visualizing environmental data on a map, providing interactive and detailed insights into the spatial distribution and trends of the data. It offers various visualization types and allows users to interactively select regions, dates, and other parameters for a customized view. As you work with or extend this script, maintain a focus on user interaction, data integrity, and the overall narrative that the visualizations convey about the data.

Key Components

1. **Imports**

The script imports necessary libraries for data handling, visualization, and map interaction:

- `numpy` and `pandas`: For data manipulation and analysis.
- `matplotlib.pyplot` and `cartopy`: For creating static plots and maps.
- `geemap` and `ee`: For integrating with Google Earth Engine and creating interactive maps.
- `ipywidgets`: For creating interactive UI elements.
- `solara`: Likely used for reactive programming patterns in the application.
- `geedim`: Possibly used for additional Earth Engine data manipulation or visualization.

2. **MapVizUI Class**

The `MapVizUI` class encapsulates all functionality related to map visualization.

- **Initialization (`__init__`)**:
  - Initializes the class with a dataframe, a flag indicating if the data is available, and optional parameters for coordinates and country.
  - Sets up the map using `geemap.Map()` and initializes various reactive variables for managing the state of the visualization.
  - Determines the type of region (point, polygon, or feature collection) based on the provided coordinates or country.
- **Date and Visualization Type Selection Components**:
  - Provides dropdown components for selecting dates, visualization types, value types, and hours.
  - Reactively updates the available options and disables/enables components based on user selections.
- **Region Type Determination (`determine_region_type`)**:
  - Determines the type of region selected for visualization based on the provided coordinates or country.
- **Map Extent Calculation (`get_map_extent`)**:
  - Calculates the map extent based on the selected region, country, or coordinates.
- **Plot Generation (`on_plot_button_clicked`)**:
  - The main function that generates the visualization based on user selections.
  - Supports different visualization types like heat maps, Earth Engine images, and contour plots.
  - Filters the dataframe based on the selected date range and visualization type, then creates and displays the map or plot.

- **CSV Download (`csv_download`)**:
  - Provides functionality to download the data as a CSV file.
- **Main UI Layout (`MapVizUILayout`)**:
  - Orchestrates the layout and rendering of the date and visualization type selection components, along with the output area for displaying the generated map or plot.
- **Display Function (`display`)**:
  - Returns the main UI layout for rendering in the application.

Development Tips

- **Understanding Map Libraries**: Familiarize yourself with the **`geemap`** library and how it integrates with Google Earth Engine for creating interactive maps.
- **Data Integrity**: Ensure that the data passed to the class is clean and well-formatted. The class expects a specific structure, particularly when setting indices and extracting date, time, and location information.
- **Reactive Integration**: The class is likely used in a reactive environment. Understand how changes in the data or user interaction might trigger re-rendering or updating of the visualizations.
- **Extending Visualizations**: If you need to add new types of visualizations or customize existing ones, focus on the plot generation function and ensure that any new data processing aligns with the expected format of the visualization library.