Solar Suitability Dashboard

Interactive visualization for solar suitability analysis

Mohammad Faiz Alam

Regional Researcher, IWMI

m.alam@cgiar.org

IWMI
International Water
Management Institute

Overview

The Solar Suitability Dashboard is an interactive web application that visualizes solar suitability across various regions. It provides stakeholders with a powerful tool to analyze multiple factors affecting solar potential and make data-driven decisions for solar implementation strategies.

This dashboard was developed for the International Water Management Institute (IWMI) to support sustainable energy and water management initiatives.

Key Features

- **Interactive Geographic Visualization**
  - Explore solar suitability across regions
  - Color-coded maps for intuitive understanding
- **Multi-layered Analysis**
  - Examine various factors affecting solar potential
  - Compare different parameters
- **Statistical Insights**
  - Distribution of suitability levels
  - Percentage breakdowns
- **Comparative Analysis**
  - Compare regions at multiple scales
  - Identify optimal areas

Technical Architecture

The dashboard is built with a modular architecture to ensure maintainability, reusability, and performance optimization.

Data Modules

- Loader: Data loading and caching
- Processor: Data filtering
- Calculator: Statistical calculations

Visualization Modules

- Maps: Geographic visualization
- Charts: Statistical visualization

UI Modules

- Layout: Page structure components
- Styles: CSS and styling

Utility Modules

- Constants: Application-wide constants
- Common functions and tools

Project Structure

```
IWMI_dashboard_2.0/
│
├── app.py                    # Main application entry point
├── modules/                  # Application modules
│   ├── data/                 # Data handling modules
│   │   ├── loader.py         # Loading and caching
│   │   ├── processor.py      # Data filtering
│   │   └── calculator.py     # Statistical calculations
│   ├── visualization/        # Visualization modules
│   │   ├── maps.py           # Map creation functions
│   │   └── charts.py         # Charts and plots
│   ├── ui/                   # UI components
│   │   ├── layout.py         # Page components
│   │   └── styles.py         # CSS and styling
│   └── utils/                # Utility functions
│       └── constants.py      # App constants
│
├── data/                     # Data files
│   └── shapefiles/           # GIS shapefiles
│
└── .streamlit/               # Streamlit configuration
```

Data Processing Pipeline

1. Data Loading & Caching

```python
@st.cache_data(ttl=3600)
def load_shapefile_data():
    """Load shapefile with caching"""
    gdf = gpd.read_file(SHAPEFILE_PATH)

    # Process and calculate averages
    calculated_data = calculate_averages(gdf)

    return calculated_data
```

2. Statistical Calculations

```python
def calculate_averages(_gdf):
    """Calculate state and national averages"""
    # Create aggregations by state and national
    # Calculate numeric metrics
    # Handle categorical classifications
    return result_gdf
```

3. Data Processing

```python
def filter_data_with_shapefile(
    _gdf, selected_state, selected_district
):
    """Filter data based on selection"""
    if selected_state == "National Average":
        filtered_gdf = _gdf[
            _gdf["NAME_1"] == "National Average"
        ]
    else:
        # Handle state/district filtering
        ...

    return filtered_gdf
```

4. Data Transformation

```python
def get_color_for_value(value):
    """Get color based on value"""
    # Apply color mapping based on
    # suitability categories or numeric ranges
    return color_hex_code
```

Visualization Techniques

Map Visualization

```
def create_simple_map(_gdf, selected_state,
                      selected_district,
                      vis_column, selected_category):
    """Create an interactive map"""
    # Create matplotlib plot
    fig, ax = plt.subplots(1, 1,
                           figsize=(10, 8))

    # Apply custom coloring for categories
    custom_colors = {
        "Very Highly Suitable": '#66CC66',
        "Highly Suitable": '#99FF99',
        "Moderately Suitable": '#FFFF99',
        "Less Suitable": '#CC0000'
    }

    # Generate appropriate legend
    # Create color-coded visualization
    return fig
```

Chart Visualization

```
def create_suitability_chart(df,
                             selected_state,
                             selected_category):
    """Create a pie chart showing
    distribution of suitability levels"""
    # Count occurrences of each level
    suitability_counts = df[
        selected_category].value_counts()

    # Create pie chart with Plotly
    fig = px.pie(
        suitability_counts,
        values='Count',
        names='Suitability Level',
        title=f'Distribution in {selected_state}'
    )

    # Style and return chart
    return fig
```

UI Components

Controls Panel

```
def create_controls(df):
    """Create the control panel"""
    # Create columns for controls
    all_controls = st.columns([1, 1, 1, 1.5, 0.5])

    # State selection
    with all_controls[0]:
        states = df["NAME_1"].unique()
        states_list = ["National Average"] +
                      sorted([str(s) for s in states])
        selected_state = st.selectbox(
            "State", states_list)

    # District, category & layer selections
    # ...

    return selected_state, selected_district,
           selected_category
```

Styling & Layout

```
def apply_css():
    """Apply custom CSS styling"""
    CSS = """
    <style>
        /* Headers styling */
        h1, .title-box {
            color: #d81b60 !important;
            font-weight: bold;
        }

        /* Panel styling */
        .panel {
            background-color: white;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
            padding: 1rem;
            border-left: 4px solid #1976d2;
        }

        /* Additional styling... */
    </style>
    """
    st.markdown(CSS, unsafe_allow_html=True)
```

User Guide (1/2)

Basic Navigation

1. **Select State and District**
    - **National Average**: View country-wide data and statistics
    - **State Level**: Select a state and "All Districts" for state-level view
    - **District Level**: Select specific state and district for detailed analysis
2. **Choose Analysis Category**
    - **Adaptation**: Suitability for adaptation strategies
    - **Mitigation**: Suitability for mitigation approaches
    - **Replacement**: Suitability for replacing existing energy sources
    - **General_SI**: General Suitability Index
3. **Select Data Layer**
    - Choose specific parameters to visualize on the map
    - Options include Solar radiance, Groundwater Development, Irrigation Coverage, etc.

User Guide (2/2)

Interpreting the Results

Map Color Coding:

- **Dark Green**: Very Highly Suitable areas
- **Light Green**: Highly Suitable areas
- **Yellow**: Moderately Suitable areas
- **Red**: Less Suitable areas
- **Gray**: Mixed or Insufficient Data

Statistics Panel Features:

- Information about selected region (state/district)
- Suitability level for selected category
- Value for selected layer parameter
- Distribution charts (for state and national views)
- Percentage distributions

Installation and Setup

Prerequisites

- Python 3.8 or higher
- pip (Python package installer)
- Virtual environment (recommended)

Setup Steps

1. **Clone the repository**

```
git clone https://github.com/IWMI/Solar-Dashboard.git
cd IWMI_dashboard_2.0
```

2. **Create virtual environment**

```
python -m venv venv
.\venv\Scripts\activate  # Windows
source venv/bin/activate  # Unix/Mac
```

3. **Install dependencies**

```
pip install -r requirements.txt
```

4. **Configure shapefile path**

   - Place shapefiles in data/shapefiles/
   - Or update the path in loader.py

5. **Run the application**

```
streamlit run app.py
```

Performance Considerations

Data Optimization

- **Shapefile Optimization**

```
from modules.data.loader import optimize_shapefile
optimize_shapefile()
```

- **Data Caching**
  - Streamlit's `st.cache_data` mechanism
  - Reduced reload frequency
  - Faster user experience

Hardware Requirements

- 4GB RAM minimum (8GB recommended)
- Modern web browser
- Internet connection (for deployment)

Common Issues

1. **Shapefile Not Found**
   - Check file path in loader.py
   - Verify all shapefile components exist
2. **Rendering Issues**
   - Update browser to latest version
   - Try a different browser
   - Ensure sufficient memory
3. **Package Dependencies**
   - Install packages individually if needed:

```
pip install streamlit pandas geopandas
matplotlib plotly
```

Future Enhancements

Data Enhancements

- Temporal data integration
- Higher resolution geospatial data
- Additional sustainability metrics

Analysis Capabilities

- Predictive modeling
- Time-series visualization
- Cost-benefit analysis integration

User Interface Improvements

- Mobile-responsive design
- Downloadable reports
- Custom area selection

Technical Improvements

- API for external system integration
- Automated data updates
- Enhanced performance optimization

Implementation Benefits

For Decision Makers

- Data-driven solar implementation
- Comprehensive spatial understanding
- Efficient resource allocation
- Regional comparison capabilities
- Better investment decisions

For Technical Users

- Modular, maintainable codebase
- Optimized for performance
- Easy to extend
- Well-documented
- Reusable components

Thank You!

Contact Information

Mohammad Faiz Alam

Regional Researcher

International Water Management Institute (IWMI)

Email: m.alam@cgiar.org

Questions?

IWMI
International Water Management Institute